

אנדריאס פילאווקיס

ללמוד UNIX



עורך יצחק עמיהוד

ספרי לימוד והכשרה
במדעי המחשב
הוצאת הוד-עמי

ניר צורים

ללמוד UNIX

עריכה: יצחק עמיהוד

תרגום: חיים פליבה

ללמוד UNIX

ספרי לימוד והכשרה במדעי המחשב
הוצאת הוד-עמי

U N I X Workshop

Andreas J. Pilavakis

ISBN 0-333-48850-4
ISBN 0-333-48851-2 Pbk

Authorized translation from
the English language edition

Hod-Ami Publishers Ltd.
P.O. Box 560, Ramat-Gan 52105
Israel

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 560 רמת-גן

הודפס בישראל
תמוז תש"ן, 1990
Printed in Israel, 1990

ISBN 965-361-00000 מסת"ב

תוכן העניינים

הקדמה

מערכת UNIX פותחה בשנות ה-70 המוקדמות במעבדות בל של חברת AT&T, על ידי קן תומפסון (Ken Thompson), דניס ריצ'י (Dennis Ritchie) ומספר מדענים אחרים. הפיתוח של UNIX הושפע ממערכת ההפעלה Multics, אשר רצה על מחשב GE645 בשנות ה-60 המאוחרות.

הגרסה המוקדמת ביותר של UNIX רצה על PDP-7 והוסבה ב-1971 ל-PDP-11. בזמן החיפוש אחר שפה ניידת (portable) למערכת ההפעלה החדשה, החליט ריצ'י לפתח את שפת C, אשר נגזרה משפת B, שנגזרה משפת BCPL. שפת B היתה שפה מפרשת (interpreter) שפותחה ע"י תומפסון.

בשנת 1973 נכתבה UNIX מחדש בשפת C; היה זה רעיון חדשני למדי באותה תקופה. לאחר מכן גדל בהרבה מספר יחידות המחשב ב-AT&T שהשתמשו ב-UNIX. אולם, מספר פסקי דין בארה"ב שהתקבלו בשנות ה-50, אסרו לשווק את UNIX לציבור. לדוגמה, ווסטרן אלקטריק (חברת השיווק של AT&T) לא הורשתה לתת תמיכה, לתחזק או לתת אחריות ל-UNIX. כתוצאה מכך, הארגונים היחידים שיכלו להנות מ-UNIX באותה תקופה היו בעיקר ארגונים ללא מטרת רווח ומוסדות חינוך, תמורת תשלום מינימלי, אשר כיסה את הוצאות הטיפול והמשלוח בלבד.

בשנת 1975 הפכה גרסה 6, שהיתה מרובת תכניות (multi-programming), לגרסה הראשונה של UNIX שהופצה לציבור הרחב. ב-1977 החל תהליך של הסבת UNIX לארכיטקטורות שלא היו מתוצרת דיגיטל, כמו אינטרדטה 8/34. גרסאות מאוחרות יותר הן גרסה 7 שהיתה זמינה ב-1978, System-III ב-1981, וגרסת System-V מ-1983 שנתמכה בצורה מלאה על ידי AT&T.

UNIX נהנתה מ"תקופת הריון" ארוכה, אשר במהלכה היתה נתונה בשליטה מלאה של מתכנניה. אנשי אקדמיה ואנשים חיצוניים אחרים אשר קיבלו את התוכנה לשימושם תרמו רבות לפיתוחה. התוצאה הסופית היתה מערכת של מאפיינים שהגדילה במידה רבה את הפופולריות של UNIX בשוק המסחרי. נציין כמה מן המאפיינים:

* מערכת הפעלה ניידת ביותר.
* מישק (interface) משתמש פשוט, אך חזק ביותר.
* מערכת קבצים היררכית קלה לתחזוקה.

* סביבת פיתוח דינמית, אשר ניתן לפתח בה תכניות ולהעביר אותן למחשבים אחרים מבלי לבצע אף לא שינוי אחד, או עם שינויים מעטים ביותר.

UNIX הינה מערכת הפעלה ניידת, רבת משימות (multitasking) ומרובת משתמשים (multiusere), אשר מספקת סביבת פיתוח ידידותית ועתירת תכונות ואפשרויות. מערכת ההפעלה נתמכת בצורה מסחרית על ידי AT&T, על ידי יצרנים חשובים של מיקרו מעבדים כמו אינטל ומוטורולה, על ידי יצרני מערכות ועל ידי מפתחי תוכנה. דבר זה הופך אותה למערכת הפעלה מובילה בשוק.

הספר מבוסס על ניסיונו של המחבר בעבודתו כמדריך, מרצה, מנתח מערכות ויועץ למחשוב.

תשעת הפרקים הראשונים מיועדים לסטודנטים ולכל אדם הרוצה לרכוש ידע שיאפשר לו לעבוד ב-UNIX. שאר פרקי הספר מכסים מספר תכניות שירות חשובות ב-UNIX, אשר משמשות לפיתוח ולתחזוקה של חבילות תוכנה. מוסברים גם מאפיינים מתקדמים של תקשורת בין תהליכים כמו אתתים (semaphores), סגמנטים משותפים ותורי הודעות (massage queues). פרקים מתקדמים אלה פונים למשתמשי UNIX אשר מעורבים בפיתוח של חבילות תוכנה ובהסבת חבילות תוכנה ממערכות הפעלה אחרות אל UNIX.

הספר מכסה מגוון רחב ביותר של נושאים הקשורים ב-UNIX. מטרה זו הושגה על ידי כתיבה תמציתית ועל ידי שימוש בדוגמאות שימושיות שונות המחליפות דפים רבים של הסברים תיאורטיים. למרות שהספר מתחיל בתיאור היסודות של UNIX לפני שהוא עוסק במאפיינים מתקדמים של מערכת ההפעלה, הוא מיועד לקורא המכיר באופן בסיסי לפחות שפת תכנות עילית אחת. הנושאים המתקדמים בספר מיועדים לקורא בעל ניסיון בתכנות בשפת C.

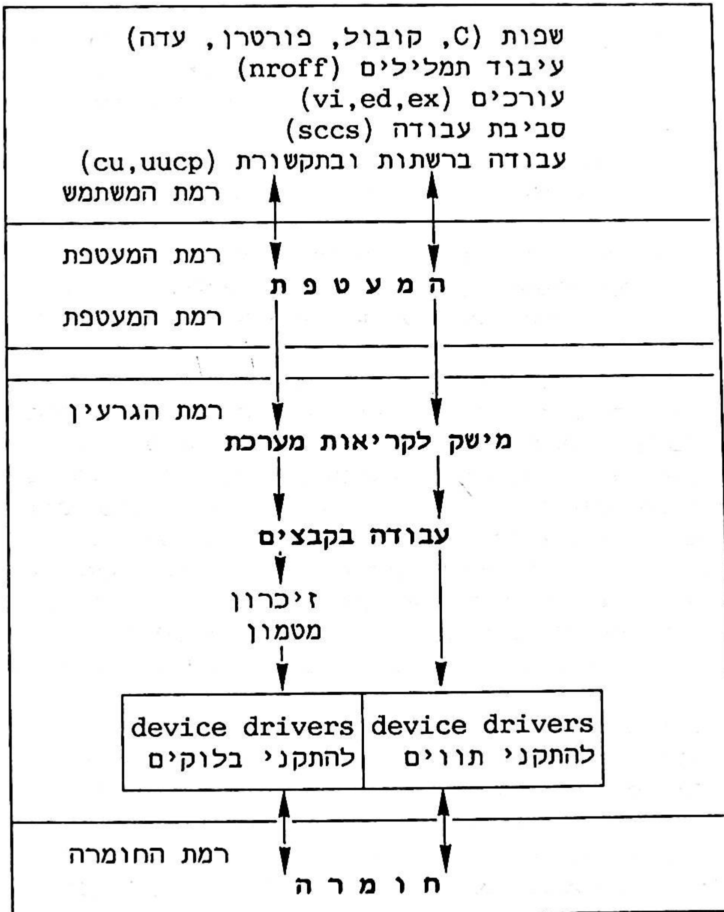
אני רוצה להודות לאחי, קירייקוס, על עזרתו והעידוד שקיבלתי ממנו בזמן כתיבת הספר, כמו גם על ההגהות אשר עשה.

ברצוני להודות גם לאנשים הבאים שקראו חלקים מהספר ותרמו הערות מועילות: ג'יימס ל. מיירס, ג'ון אפלבי, פיטר טינטל, מרטין סוואש, ז'אן פייר ברטינצ'אמפ.

פרק 1

סקירה על מערכת UNIX

פרק זה הינו סקירה על מערכת ההפעלה UNIX. הוא כולל גם מידע על תהליך הכניסה (login) למערכת, הסבר על תווים מיוחדים ועל תהליך היציאה (logout).



תרשים 1.1 **תיאור סכימטי של מבנה UNIX**

כפי שניתן לראות מתרשים 1.1 המשתמש יכול לשוחח עם גרעין ה-UNIX (UNIX Kernel) באמצעות מספר שפות תכנות ותכניות שירות. תכניות כמו המעטפת (shell) ו-SCCS משתמשות בשירותים שמספק הגרעין באמצעות מערכת מוגדרת מראש של קריאות מערכת (system calls). קריאות מערכת אלה מיועדות, לדוגמה, ליצירת קבצים חדשים, לכניסה לקבצים קיימים, להתקנת צינורות (pipes) ולהתקנה והסרה של מערכות קבצים (file systems). כל המונחים החדשים האלה יוסברו בפרקים הבאים.

1.1 הגרעין

הגרעין (kernel) הינו הבסיס של מערכת ההפעלה. זוהי תוכנה הנמצאת בזיכרון ותומכת בסביבה מרובת משתמשים ורבת המשימות של UNIX. הוא מתבסס על כ-13,000 שורות הכתובות בשפת C ועל כ-1,000 שורות הכתובות באסמבלי.

הגרעין אחראי לניהול התהליכים. תהליך (process) הינו תכנית ביצוע בצירוף מבני הנתונים שלה. תהליך (הנקרא "אב" - parent) יכול ליצור תהליכי משנה נוספים ("ילדים" - children). הגרעין מבטיח שהילדים יירשו את סביבת האב. התקשורת בין תהליכים, אשר מנוהלת על ידי הגרעין, מאפשרת לתהליכים להעביר הודעות זה לזה וליצור קשר באמצעות אתתים.

כדי להקל על הדרישה לזיכרון המרכזי, מועברים התהליכים מהזיכרון לדיסק וחזרה לזיכרון בעזרת אלגוריתם יעיל. באופן רגיל נעשה שימוש בשתי שיטות כדי לנהל תהליכים: העברה (swapping) ודפדוף (paging). בתהליך ההעברה, מועברים תהליכים שלמים בין הזיכרון המרכזי לבין ההתקן המשמש לאחסון התהליכים. בתהליך הדפדוף, מועברים דפים בודדים של התהליך לדיסק וממנו, לפי צורכי התהליך באותו רגע. היתרון של דפדוף על העברות נעוץ בכך שהוא מאפשר ליותר תהליכים להשתמש בזיכרון המרכזי. הדפדוף גם מאפשר לתהליך להיות גדול יותר מאשר הזיכרון המרכזי העומד לרשותו.

הגרעין מפקח גם על שידור הנתונים בין הזיכרון המרכזי ליחידות ההיקפיות. חלק זה של הגרעין שנוה במידה רבה במערכות ההפעלה UNIX הפועלות במחשבים של יצרנים שונים, מכיון שהציוד ההיקפי ותכונותיו משתנה ממחשב אחד למשנהו.

כאשר תכנית נטענת לזיכרון לצורך ביצוע, נטענים למעשה התמליל (הפקודות - text), הנתונים והסגמנטים הלוגיים של המחסנית שלה. החלק של הגרעין העוסק בניהול הזיכרון מאפשר שיתוף בסגמנט התמליל של התוכנית וגם הרחבה דינמית של הסגמנטים

המכילים נתונים, או אלה המשמשים את המחסנית.

תזמון תהליכים	תחזוקת מערכת קבצים
הקצאה של משאבי מערכת	ניהול זיכרון

תרשים 1.2 מבט על הגרעין

1.2 המעטפת

המעטפת (shell) הינה תכנית מערכת המאפשרת תקשורת דו-כיוונית בין המשתמש למחשב. היא פועלת בצורה הדברותית כמפרשת של פקודות (command interpreter). היא גם שפת תכנות, עם משתנים המוגדרים על ידי המשתמש ועם אופציות לבקרת זרימה (control flow).

אופן הפעולה של המעטפת הינו מורכב למדי. בתחילה, מתורגמת הפקודה או ההוראה של המשתמש והתכנית המתאימה, המייצגת את אותה פקודה, מאותרת בדיסק. לאחר מכן נוצר תהליך אשר מבצע את התכנית והמעטפת ממתינה עד אשר התכנית מסתיימת. לבסוף, היא מנחה (prompt) את המשתמש להכניס את הפקודה הבאה. על ידי סיום פקודה בסימן "&" יכול המשתמש לבצע פקודות ברקע (background) כלומר, המעטפת תיצור שוב תהליך כדי לבצע את הפקודה, אבל היא לא תמתין לסיומו ותקבל את הפקודה הבאה.

עקרון העבודה של צינורות (pipes) מאפשר למעטפת ליצור מספר תהליכים שיכולים להתקשר ביניהם. באופן כלל, אם נדרש טיפול נוסף לפלט של תהליך, הוא מועבר דרך הצינור לקלט הסטנדרטי של תהליך אחר. תפישת העבודה של תכניות הסינון (filters), שבה תכנית קוראת נתונים מאמצעי קלט סטנדרטי, משנה אותם (למעשה הנתונים "מסוננים" על ידי התכנית) וכותבת אותם לאמצעי פלט סטנדרטי, נתמכת גם היא ע"י הגרעין.

שפת התכנות של המעטפת מאפשרת למשתמש להגדיר משתנים ולהשתמש במבנים הדומים לאלה של שפת תכנות, כדי ליצור רצפי פקודות מורכבים. רצף פקודות מאוחסן נקרא "קובץ פקודות", אך הוא ידוע יותר בשם shell script, או בקיצור script (בהמשך נשתמש במונח "קובץ פקודות" - command file). ניתן להשתמש בקבצי פקודות כדי לגרום לביצוע סדרתי של רבות מתכניות השירות של

UNIX ועל ידי כך להגביר במידה רבה את עוצמתה של מערכת ההפעלה.

שפת פקודות גמישה	שרשור פקודות וניתוב ק/פ
ביצוע ב-foreground וב-background	צינורות ומסננים

תרשים 1.3 מבט על המעטפת

1.3 פרוטוקול הכניסה

כדי להכנס (login) למערכת UNIX, עליך להשיג ממנהל המערכת שלך מספר מזהה (username) וסיסמה (password). מסוף UNIX ינחה אותך עם ההודעה:

```
login:
```

כתגובה להודעת "login", הקש את המספר המזהה שלך באותיות קטנות ולאחריו הקש על מקש "Return" (CR). מערכת UNIX תדפיס:

```
password:
```

הכנס את הסיסמה שלך ולאחריה הקש CR. אם כל הפרטים נכונים תכנס למערכת. אם לא כן, תצטרך לחזור על כל השלבים של הכנסת המספר המזהה והסיסמה.

1.4 לאחר הכניסה למערכת

לאחר שנכנסת, אתה נמצא בקשר עם מפרש הפקודות של המעטפת. אתה יכול לשנות את הסיסמה שלך על ידי שימוש בפקודה:

```
$passwd
```

שים לב שהסימן "$" בדוגמה לעיל אינו חלק מהפקודה. למעשה זהו מנחה ה-UNIX, אשר יכול להיות שונה בכל יחידת מחשב המשתמשת ב-UNIX. בספר זה נשתמש בסימן "$" כדי להקדים את כל פקודות UNIX.

מספר תווים, אשר אינם בהכרח זהים בכל מערכות UNIX, מטופלים בצורה מיוחדת ברמת הפקודה של המעטפת. להלן מקצת מבין התווים האלה ומשמעותם:

* # - מחיקה של התו האחרון שהוקלד.
* הקשה על CONTROL-U - מחיקת שורה שלמה שהוקלדה.
* הקשה על CONTROL-D - הדמייה של תו סוף קובץ.
* הקשה על DEL או RUB - מפסיקה פקודה (interrupt).
* הקשה על Return (CR) - סוף הקלט, הוראה לבצע את הפקודה.

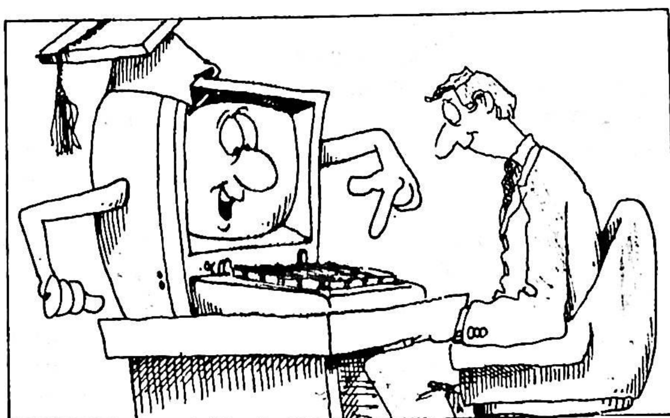
כדי לקבל מידע על תכניות השירות של UNIX או על פקודות, אתה יכול להשתמש בפקודת **man** לדוגמה:

```
$man passwd
```

פקודה זו תציג מידע על האופן שבו יש להשתמש בפקודת **passwd** כדי לשנות את הסיסמה.

לאחר שסיימת את השימוש במסוף, אתה יכול לצאת על ידי הקשה על מקשי CONTROL-D, או על ידי הקלדה של הפקודה הבאה:

```
$logout
```



פרק 2

מערכת הקבצים ב-UNIX

פרק זה מתאר את המבנה הבסיסי של מערכת הקבצים ב-UNIX ואת סוגי הקבצים שניתן להשתמש בהם. בנוסף לכך נתאר מספר פקודות פשוטות אך חיוניות:

pwd	cd	mkdir
rmdir	rm	ls
cat	pg	lp
pr	cp	mv
find	ln	chmod
umask	mount	umount

2.1 קבצי UNIX

מערכת ההפעלה UNIX משתמשת במערכת קבצים היררכית, המאורגנת כעץ. הצומת הראשי נקרא מדריך ראשי (root directory), אשר מיוצג על ידי סימן יחיד "/" (slash). למשתמש יש גישה לארבעה סוגים של קבצים:

* **קבצים רגילים** אשר נוצרים על ידי המשתמש (תכניות C, דוחות, קבצי פקודות של המעטפת וכו').
* **מדריכים** אשר מתנהגים כמו קבצים רגילים, אך רק משתמשים בעלי הרשאה מתאימה יכולים לכתוב בהם. מדריכים משמשים כדי לקבץ קבצים שיש קשר ביניהם וכדי לקבוע זיקה (mapping) בין שמות קבצים לוגיים לשמות פיסיים.
* **צינורות** (pipes), אשר משמשים לקישור פלט של תהליך אחד לקלט של תהליך אחר. הם מותקנים בגרעין כמאגרים הפועלים לפי שיטת ראשון-נכנס-ראשון-יוצא (FIFO). גירסת UNIX-V מבדילה בין שני סוגים של צינורות: צינורות ללא שמות (הנוצרים על ידי קריאת המערכת **pipe** וצינורות עם שמות (הנוצרים על ידי קריאת המערכת **mknod** ומשמשים לתקשורת בין תהליכים שאינם קשורים זה לזה).
* **קבצים מיוחדים**, אשר משמשים את המערכת לביצוע פעולות מנהלה ובקרה. קבצים מיוחדים אלה משמשים למשל, כדי לפקח על התקני קלט/פלט (ראה סעיף 2.18).

/

unix bin etc usr dev

kernel ls mv tty05

rony andy

swap dat

מקרא:

- מדריך

- קובץ רגיל

- קובץ מיוחד

תרשים 2.1 קבצים בעץ טיפוסי של מערכת קבצים

שם קובץ במדריך יכול להכיל עד 14 תווים. בשמות קבצים לא ניתן להשתמש בתווים הבאים: ; & / $

כל מדריך מכיל לפחות שתי רשומות; רשומת "." אשר מתייחסת למדריך עצמו ורשומת ".." אשר מתייחסת לאב של המדריך הפעיל הנוכחי. ניתן לראות רשומות אלו בעזרת הפקודה **ls -al**). להלן כמה מבין המדריכים המנהליים:

* / - מדריך ראשי
* /unix - גרעין מערכת UNIX
* /bin - פקודות
* /etc - קובצי מנהלה
* /lib - ספריות
* /tmp - קבצים זמניים

ניתן למצוא כל קובץ במערכת הקבצים על ידי שימוש במסלול (path) מהמדריך הראשי (ראה פקודת **cd**).

2.2 קביעת המדריך הנוכחי

המדריך הנוכחי הינו המדריך שבו המשתמש פועל כרגע. פקודת **pwd** משמשת למציאת המסלול המלא של המדריך הנוכחי. בהתייחס לתרשים 2.1, אם המדריך הנוכחי הוא "rony", אז הפקודה

```
$pwd
```

תציג

```
/usr/rony
```

2.3 שינוי המדריך

ניתן להשתמש בפקודה **cd** כדי לשנות את המדריך הנוכחי למדריך הרצוי. הפקודה נראית כך:

```
$cd [שם-מדריך]
```

כדי להגיע למדריך "/usr/rony" (תרשים 2.1) אתה צריך להשתמש בפקודה:

```
$cd /usr/rony
```

אם ברצונך לעבור לאחר מכן מ-"/usr/rony" ל-"/usr/andy", אתה יכול להשתמש בפקודה:

```
$cd /usr/andy
```

או בפקודה:

```
$cd ../andy
```

אפשר לכתוב פקודה זו, מכיון ש-".." מתייחס למדריך האב של "rony", שהוא "/usr".

באופן דומה, אם ברצונך לבצע את התכנית "swap" (תרשים 2.1) כאשר המדריך הנוכחי הוא "/usr/andy", אתה יכול להשתמש באחת משתי האפשרויות הבאות:

```
$/usr/rony/swap
```

או

```
$../rony/swap
```

אם לא מצויין שם מדריך כמשתנה בפקודה **cd**, הפקודה **cd** תשנה את המדריך למדריך "הבית" (HOME directory) של המשתמש. לרוב, זהו המדריך שבאמצעותו המשתמש נכנס למערכת.

2.4 יצירת מדריך חדש

פקודת **mkdir** משמשת ליצירת מדריך חדש או מדריכים חדשים. הפקודה נראית כך:

```
$mkdir [שם-מדריך/ים]
```

המדריך הנוכחי הוא "/usr/rony", כדי ליצור מדריך חדש "bin" ב-"/usr/rony", ניתן להשתמש באחת משתי האפשרויות:

```
$mkdir bin
```

או

```
$mkdir /usr/rony/bin
```

2.5 ביטול מדריך

ניתן להשתמש בפקודת **rmdir** כדי לבטל מדריכים:

```
$rmdir [שם-מדריך/ים]
```

ניתן לבטל מדריכים ריקים בלבד (כלומר, מדריך המכיל את הרשומות "." ו-".." בלבד). יתר על כן, המשתמש יכול לבטל רק מדריכים שנותנים לו הרשאת כתיבה.

2.6 ביטול קבצים

ניתן להשתמש בפקודת **rm** כדי לבטל קבצים:

```
$mr [אופציות] שם-קובצ/ים
```

שתיים מבין האופציות הנפוצות ביותר הן:

* -r לביטול רקורסיבי של כל הקבצים ומדריכי המשנה.
* -f לביטול של כל הקבצים שאין להם הרשאת כתיבה.

כדי לבטל את הקבצים "swap.c" ו-"master.o", ניתן להשתמש בפקודה:

```
$rm swap.c master.o
```

2.7 פירוט תוכן המדריך

הפקודה **ls** משמשת לפירוט תוכן המדריך:

```
ls [אופציות] [שמות קבצים]
```

להלן כמה מבין האופציות השימושיות ביותר:

- a- פרט את כל הקבצים במדריכים המצוינים בפקודה, כולל קבצים נסתרים המתחילים בנקודה.
- l- פרט בפורמט ארוך את כל הקבצים המצוינים בפקודה.
- t- פרט קבצים שמוינו לפי תאריך עדכון אחרון (הקבצים שהשתנו לאחרונה יופיעו ראשונים).
- i- פרט את מספר הצומת i (node number) בטור הראשון (ראה סעיף 2.14).

דוגמאות:

פרט את כל הקבצים המסתיימים ב-".c":

```
$ls *.c
```

פרט את כל הקבצים המתחילים באותיות גדולות:

```
$ls [A-Z]*
```

2.8 שרשור והצגה של קבצים

פקודת **cat** משרשרת את הקבצים ומציגה את תוכנם מבלי לערוך את המידע:

```
$cat [אופציות] [שמות-קבצים]
```

לדוגמה, כדי להציג את תוכן הקבצים "prog.c" ו-"swap.c", השתמש בפקודה הבאה:

```
$cat prog.c swap.c
```

2.9 הצגת תוכן הקובץ

הפקודה **pg** מאפשרת לבחון את תוכן הקובץ, מסך אחר מסך:

```
$pg [אופציות] [שמות-קבצים]
```

אם תקיש CR, יוצג מסך נוסף של תמליל (כלומר, פקודת תכנית). אם תקיש "?" לאחר שהוצג הדף הראשון, יוצגו על המסך כל האופציות האפשריות. הפקודה **pg** ממלאת את אותה פונקציה כמו הפקודה **more**, אך ההבדל העיקרי ביניהן הוא באופציות העומדות לרשותך בכל פקודה.

2.10 הדפסת קובץ

הפקודה **lp** (או **lpr** בגרסאות אחרות) משמשת להדפסת קובץ:

```
$lp [אופציות] [שמות-קבצים]
```

בדרך כלל משתמשים בפקודה **lp** יחד עם הפקודה **pr** שמדפיסה קבצים במסוף. הסיבה לכך היא שפקודת **pr** יכולה לערוך קובץ בטורים, לפי גודל דף וכו'. לדוגמה:

```
$pr -d -172 master.c ¦ lp -n2
```

פקודה זו תגרום לכך שהפלט של הקובץ "master.c" שיודפס במדפסת שורה, יהיה בעל התכונות הבאות:

* רווחים כפולים (אופציה d-) בין השורות,
* גודל דף יהיה 72 שורות (אופציה l-),
* מספר העותקים שיודפסו יהיה 2.

בגרסת UNIX-V נשתמש בפקודה **lpstat -t** כדי לראות את הסטטוס של המדפסות.

2.11 העתקת קובץ

ניתן להשתמש בפקודה **cp** כדי להעתיק קובץ אחד לקובץ אחר, או כדי להעתיק קובץ אחד או יותר למדריך.

דוגמאות:
העתק את תוכן "filea" ל-"fileb". אם "fileb" קיים, הקובץ החדש ייכתב עליו:

```
$cp filea fileb
```

העתק את תוכן "filea" ו-"filez" למדריך "/usr/rony":

```
$cp filea filez /usr/rony
```

2.12 העברת קובץ

הפקודה **mv** משמשת להעברת קבצים ע"י שינוי שמם (renaming). פקודה זו מתנהגת בצורה דומה לפקודה **cp**, פרט לכך שהיא גם מעבירה קבצים. ברוב המקרים, אם הקובץ מועבר לאותה מערכת קבצים, אין העברה פיסית, אלא רק שינויי מדריכים.

דוגמאות:
העבר את התוכן של "filea" ל-"fileb" (קובץ "filea" מבוטל):

```
$mv filea fileb
```

העבר את המדריך "dira" למדריך "dirb" (ההעברה תבוצע אם "dira" אינו המדריך הנוכחי ואם שני המדריכים הם בני אותו אב):

```
$mv dira dirb
```

2.13 חיפוש קובץ במערכת הקבצים

הפקודה **find** משמשת לאיתור קבצים בעצי המשנה של מערכת הקבצים לפי קריטריונים מסוימים. בדוגמאות שלהלן ניתן לראות כמה מבין השימושים הנפוצים ביותר של פקודת **find**:

דוגמאות:
חפש את הקובץ **"core"** (אופציה -name) במדריכים "/usr/rony" ו-"/usr/andreas" ולאחר שמצאת אותו הצג (אופציה **-print**) את שם המסלול שלו:

```
$find /usr/rony /usr/andreas -name core -print
```

הצג את שמות המסלולים של כל הקבצים במדריך הנוכחי ובמדריכי המשנה שלו:

```
$find . -print
```

הצג את שמות המסלולים של כל הקבצים במערכת שלא נכנסו אליהם (אופציה **-atime**) ולא כתבו בהם (אופציה **-mtime**) במשך 60 הימים האחרונים:

```
$find / -atime +60 -mtime +60 -print
```

2.14 קישור קבצים

לכל קובץ הרשום במדריך יש מספר זיהוי ייחודי במערכת הקבצים של UNIX, הנקרא מספר הצומת i ‏(i-node number). ניתן לייחס מספר קבצים אל קובץ פיסי אחד על ידי הצבה של אותו מספר צומת i בכניסות המתאימות של המדריך שלהם (ראה פרק 14). קישורים כאלה יכולים להיווצר על ידי שימוש בפקודה **ln** (הערה:הסוגריים נועדו להפריד בין הפרמטרים, אך אין הם חלק מהפקודה):

```
$ln (שמות קבצים) (קובץ או מדריך המטרה)
```

דוגמאות:
קשר את קובץ "filea" במדריך הנוכחי אל "fileb" שבמדריך ההעברות (swaps). לאחר הקישור, הן "filea" והן "fileb" יתייחסו לאותו קובץ פיסי בדיסק:

```
$ln filea swaps/fileb
```

קשר את "filea", "fileb" ו-"filez" על ידי אותו שם בקובץ ההעברות:

```
$ln filea fileb filez swaps
```

לא ניתן לקשר קובץ לעצמו, ואין זה חוקי לקשור קבצים בין מערכות קבצים הנמצאות בהתקני אחסון שונים.

2.15 אופני ההגנה של מערכת הקבצים

מערכת UNIX מספקת אמצעים שבעזרתם המשתמשים יכולים ליצור קבצים, להשתמש בהם במשותף ולבטל אותם, בתנאי שיש להם את ההרשאה המתאימה. המערכת מבדילה בין שלושה סוגי משתמשים:

הבעלים - המשתמש שיצר את הקובץ.
הקבוצה - הקבוצה שאליה משתייך בעלי הקובץ.
אחרים - כל שאר המשתמשים המורשים במערכת.

לקובץ השייך למשתמש מסוים יכולות להיות הרשאות הכניסה הבאות:

(r) read אפשרות לקרוא את הקובץ.
(w) write אפשרות לכתוב ולשנות את הקובץ.
(x) execute אפשרות להריץ את הקובץ..

הבה נבחן עכשיו את הפלט של הפקודה ls:

```
$ls -l
drwxrwxr-x  1  andy  usr  120   Jan  12  10:05  master
-rw-rw-r--  1  andy  usr  978   Jan  14  12:58  swap.c
-rw-rw----  1  andy  usr  2058  Jan  16  10:20  swap
```

הערה: שים לב לכך שלאחר הצגת הפקודה ניתנות שורות פלט לדוגמה. כך נעשה גם בהמשך, נציג פקודה ודוגמת פלט.

התו הראשון ברשימה הארוכה לעיל מציין את סוג הקובץ, אשר יכול להיות אחד מאלה:

d קובץ מדריך.
c קובץ מיוחד לתווים.
b קובץ מיוחד לבלוקים.
p קובץ מיוחד לצינור עם שם.
- קובץ רגיל.

תשעת התווים הבאים מציינים את הרשאות הכניסה לקובץ. תשעת תווי מערכת אלה מחולקים לשלוש קבוצות בנות שלושה תווים כל אחת. הקבוצה הראשונה של שלושה תווים מציינת את הרשאות הכניסה של בעלי הקובץ, הקבוצה השנייה - לקבוצת המשתמשים והשלישית - לאחרים.

לפיכך, אנו יכולים לראות מהפלט של פקודת **ls -l** שהקובץ "master" הינו מדריך אשר לבעליו, "andy", יש הרשאות קריאה (r), כתיבה (w), וביצוע (x); לקבוצה "usr" יש הרשאות קריאה, כתיבה וביצוע; ולאחרים יש הרשאות קריאה וביצוע בלבד.

ניתן לשנות את הרשאות הכניסה לקובץ בעזרת הפקודה **chmod**:

```
chmod אופציות שמות-קבצים
```

האופציות יכולות להיות ייצוג אוקטלי או סימבולי של ההרשאות. בשיטת הייצוג הסימבולי משתמשים בקודים הבאים:

```
[ugoa] [+ -] [rwxst]
```

הקודים של המשתמש [ugoa] הם:

u הבעלים
g הקבוצה
o האחרים
a כולם (a הינה ברירת המחדל, אשר כוללת את u, g ו-o)

קודי הפעולה [- +] הם:

+ הוסף הרשאה
- בטל הרשאה

קודי הרשאות הכניסה [rwxst] הם:

r קריאה
w כתיבה
x ביצוע
s סיבית מספר מזהה (uid bit)
t סיבית צמודה (sticky bit)

דוגמאות:
כדי לשנות את הרשאות הכניסה לקובץ ההעברות (swap) כך שלבעלים, לקבוצה ולכל האחרים תהיינה הרשאות קריאה, כתיבה וביצוע, אפשר להשתמש בקוד 777. הסיבה לכך היא שהייצוג האוקטלי של 777 הינו 111 111 111 (בהתאמה rwxrwxrwx).

```
$chmod 777 swap
```

כדי לשנות את הרשאות הכניסה לקובץ ההעברות, כך שלבעלים ולקבוצה תהיינה הרשאת כתיבה וקריאה ולאחרים תהיה הרשאת קריאה בלבד אפשר להשתמש בקוד 664. הסיבה לכך היא שהייצוג האוקטלי של 664 הינו 100 110 110 (בהתאמה --rw-rw-r).

```
$chmod 664 swap
```

כאשר סיבית הזיהוי של המשתמש (SUID - Set User ID), נקבעת ("נדלקת") בקובץ הניתן לביצוע, המערכת נותנת למשתמש המבצע את הקובץ אותן הרשאות כמו לבעלי הקובץ. באופן דומה, סיבית הזיהוי של הקבוצה (SGID - Set Group ID) מאפשרת למשתמש להשיג את אותן הרשאות כמו לחברי הקבוצה שאליה משתייך בעלי הקובץ. התשובה לשאלה איך ניתן לבצע זאת טמונה בעובדה שב-UNIX לכל תהליך יש שני UID ושני GID. הזוג הראשון של ה-UID וה-GID, הידוע בשם הזוג "האמיתי", משמש לזיהוי הבעלים של התהליך ולזיהוי הקבוצה שאליה הוא משתייך. הזוג השני, הידוע בשם הזוג "האפקטיבי", משמש לבדיקת הרשאות הכניסה לקובץ המאוחסן.

ברוב המקרים מספרי הזיהוי האפקטיביים והאמיתיים יהיו זהים. אולם כאשר תהליך מבצע תכנית אשר נקבעו בה SUID או SGID, הופכים מספרי הזיהוי האפקטיביים שלו להיות זהים לאלה של בעלי התכנית ו/או הקבוצה שאליה שייך בעלי התכנית. בצורה זו יכול התהליך שקרא להכנס לקבצים ולעדכן אותם בעזרת התכנית שנקראה.

לאחר קביעת ("הדלקת") SUID, משתנה התו המתיר לבעלים לבצע את הקובץ, מ-"x" ל-"s". פקודה טיפוסית ב-UNIX שנקבע בה SUID היא **passwd**, אשר מאפשרת למשתמשים רגילים לשנות את הסיסמה שלהם. פקודה זו מעדכנת את קובץ "/etc/passwd" שבו יש לפקודת **passwd** הרשאות כתיבה וקריאה ולמשתמשים רגילים יש הרשאת קריאה בלבד.

הסיבית הצמודה נקבעת בדרך כלל על ידי מנהלן המערכת לפקודות שהשימוש בהן הוא בתדירות גבוהה בלבד, כמו **ls** או **vi**. ברשומת הקובץ היא מצוינת על ידי התו "t" המסמל את סוג הקובץ. לאחר

שנקבעת הסיבית הצמודה, מאוחסן באופן קבוע עותק של התכנית הניתנת לביצוע (executable), בשטח הדיסק הידוע כמרחב הכתובות המשמש להעברות (swap space).

שלא כמו שאר מרחבי הכתובות בדיסק, מאורגן מרחב הכתובות המשמש להעברות באופן סדרתי. בצורה זו מצטמצם באופן משמעותי הזמן שנדרש לטעון ולבצע תכנית מתוך קובץ ההעברות כי התכנית נטענת לזיכרון כיחידה אחת ולא בלוק אחד בכל קריאה לדיסק.

צריך לזכור, שהסיבית הצמודה גורמת לכך שתכניות הנמצאות בשימוש לעתים קרובות תשארנה בשטח אחסון שניתן להכנס אליו במהירות. יש להשתמש בסיבית הצמודה בתשומת לב, מכיון שמרחב הכתובות להעברות משמש לתכניות שהביצוע שלהן הושעה באופן זמני והוא מוגבל בגודלו (במיוחד במערכות קטנות).

דוגמאות נוספות:

קבע את סיבית זיהוי של המשתמש - שם קובץ	$chmod u+s
קבע את סיבית הזיהוי של המשתמש - שם קובץ	$chmod 4000
קבע את סיבית הזיהוי של הקבוצה - שם קובץ	$chmod g+s
קבע את סיבית הזיהוי של הקבוצה - שם קובץ	$chmod 6000
קבע את הסיבית הצמודה - שם קובץ	$chmod + t

2.16 ברירת מחדל בעת יצירת קובץ

כאשר נוצר קובץ או מדריך, מקצה לו המערכת את ברירת המחדל של הרשאות הכניסה (mask), אשר לרוב ערכה האוקטלי הוא 666 (קריאה וכתיבה על ידי הבעלים, הקבוצה והאחרים). ניתן לשנות את ברירת המחדל בעזרת הפקודה **umask**:

$umask [אופציות]

המשתנה של **umask** מופחת מערך ברירת המחדל. בצורה זו תגרום הפקודה **"$umask 022"** שהקבצים ייווצרו עם הרשאות כניסה 644, מכיון ש-666 פחות 022 שווה ל-644. אם נשתמש בפקודה זו ללא משתנה, היא תציג את הערך הנוכחי של ברירת המחדל. מנהלני המערכת נוהגים לקבוע את הערך של **umask** ב-"/etc/profile".

2.17 מערכת הקבצים ב-UNIX

כל מערכת קבצים מתבססת על ארבעה חלקים בסיסיים: בלוק התיחול, סופרבלוק, בלוק רשימת-I ובלוקים המשמשים לנתונים. הבלוקים במערכת הקבצים ממוספרים מאפס ועד לגבול מערכת הקבצים.

מערכת הקבצים מאוחסנת בהתקן פיסי (דיסק), אשר יכול להיות מחולק למספר התקנים לוגיים (מחיצות). ניתן לאחסן מערכות קבצים בהתקנים לוגיים ולכן יכול התקן פיסי אחד להכיל מספר מערכות קבצים.

מערכת הקבצים הראשית (root) נמצאת בראש ההירככיה של הקבצים ולא ניתן לבטל אותה (ראה פקודות **mount** ו-**umount**), אך ניתן לבטל מערכות קבצים אחרות. קבצים במערכת קבצים הניתנת לביטול יהיו זמינים רק לאחר שמערכת הקבצים הותקנה, או קושרה (mount) לוגית,למערכת UNIX. תיאור מפורט יותר של מערכת הקבצים מופיע ניתן בפרק 14.

2.18 קבצים מיוחדים והתקנים

קבצים מיוחדים מאוחסנים בדרך כלל במדריך "/dev". קבצים אלה אינם מכילים נתונים והם משמשים לכניסה ליחידות היקפיות במערכת כמו מסופים, כונני דיסקים, כונני סרטים ומדפסות. כל אחד מהתקנים אלה מופעל באמצעות מודול של תוכנה הקרוי device driver. בקשה לקריאה או כתיבה לקובץ מיוחד גורמת להפעלה של device driver מתאים להתקן הפיסי. הבה נתבונן בפלט של הפקודה הבאה:

```
$ls -al /dev
crw--w--w-   3  root   2,  0  Mar  11  16:49  /dev/console
brw-rw-rw-   2  root   2,  6  May  19  11:23  /dev/fd70
```

התו הראשון ברשומה מייצג את סוג ההתקן המזוהה עם ההתקן הפיסי:

b מציין התקן בלוקים (כמו כונן דיסקים).
c מציין התקן תווים (כמו מדפסת, או מסוף).

המספרים הבאים לאחר הקבוצה הראשונה מציינים את מספר ההתקן העיקרי (major) והמשני (minor). המספר העיקרי מציין באיזה device driver (לדוגמה, סרט מגנטי) יש להשתמש וערכו הוא אינדקס לטבלת device drivers. המספר המשני מועבר כארגומנט ל-device driver; הוא מפרט מידע התלוי בהתקן, כמו מספר כניסה בכרטיס קלט/פלט, האם יש לגולל סרט מגנטי להתחלה בזמן הסגירה, מידע על המחיצות בדיסק וכן הלאה.

2.19 התקנה, או קישור של מערכת קבצים

יש לקשר מערכת קבצים ל-UNIX לפני שניתן להשתמש בה. עושים זאת בשני שלבים:

- יצירת מדריך ריק במדריך הראשי.
- שימוש בפקודת **mount** כדי לקשר את מערכת הקבצים להתקן לוגי.

לדוגמה, הפקודות

```
$cd /
$mkdir /master
/etc/mount /dev/fd50 /master
```

יקשרו את מערכת הקבצים על ההתקן "/dev/fd50" במדריך "/master". כך, גרעין UNIX יידע על קיומה של מערכת הקבצים החדשה. על ידי שינוי של המדריך ל-"/master", ניתן יהיה ליצור קבצים, למחוק ולשנות אותם כאילו הם היו מוגדרים בהתקן הראשי. בכל התקנה של מערכת קבצים, מעודכן קובץ בדיסק (בדרך כלל "/etc/mnttab") עם רשומה של מערכת הקבצים החדשה. ניתן להתקין מערכת קבצים עם הרשאת קריאה בלבד על ידי שימוש באופציה **r-**:

```
/etc/mount /dev/fd50 /master -r
```

שימוש בפקודה **mount** ללא ארגומנטים יגרום לכך שכל מערכות הקבצים המותקנות במערכת תוצגנה במסך:

```
$/etc/mount
/ on /dev/dsk/0s1 read/write on Mon Dec 19 17:53:20 1988
/usr/acct on /dev/dsk/1s1 read/write on Mon Dec 19 17:56:34 1988
```

2.20 הורדה של מערכת קבצים

ניתן לבטל (unmount) מערכת קבצים בעזרת הפקודה **umount**. בזמן הביטול עלולה להופיע הודעת השגיאה הבאה:

```
mount device busy
```

ההודעה הזו מוצגת כאשר מישהו משתמש במדריך של הדיסק שאתה רוצה לבטל, או כאשר מישהו פתח קובץ באותו התקן. לא ניתן לבטל מערכת קבצים ראשית ("/dev/dsk/0s1" בדוגמה לעיל). לדוגמה, כדי לבטל את מערכת הקבצים "/dev/dsk/1s1" נכתוב:

```
$/etc/mount /dev/dsk/1s1
```

פרק 3

מעטפת בורן

מעטפת בורן הינה מפרש פקודות הדברותי. היא גם שפת תכנות עם משתנים שניתנים להגדרה על ידי המשתמש וכוללת אפשרויות פיקוח על זרימה.

פרק זה נועד לתאר את מעטפת בורן (Bourne shell) על מנת לאפשר למשתמש ב-UNIX להשתמש בה ביעילות ובאמינות. מאפייני המעטפת כוללים עיבוד בחזית (foreground) וברקע (background), ניתוב קלט/פלט סטנדרטי, צינורות ומסננים (פילטרים), קובצי פקודות, תווים שמורים ומשתני המעטפת.

3.1 פקודות

המעטפת קוראת פקודות ממסוף או מקובץ ומבצעת אותן. פקודה מורכבת מרצף של מלים המופרדות ביניהן על ידי מרווחים. המילה הראשונה מציינת את הפקודה (אשר יכולה להיות תכנית או קובץ פקודות) שיש לבצע. לדוגמה, הפקודה **who** תציג את המשתמשים אשר מחוברים כרגע למערכת:

```
$who
andreas   tty01 Dec 12 08:30
paul      tty05 Dec 12 09:05
```

הערה: שים לב, אנו מציגים פקודה ומתחתה - דוגמת פלט אפשרי.

כדי לזהות את המשתמש כרגע במסוף מסוים, הקש:

```
$who am I
andreas  tty01 Dec 12 10:31
```

יש לסיים כל פקודה בהקשת "Return". אפשר להקיש מספר פקודות בשורה אחת על ידי הפרדה בנקודה-פסיק (;):

```
$who am I;date
andreas  tty01 Dec 12 10:32
Mon Dec 12 10:32:21 MST 1988
```

3.2 דגלים

דגלים (flag arguments) מציינים הפעלה של אופציות המותרות לפקודה מסוימת. לדוגמה, הפקודה ls ללא דגל תציג את הפרטים הבאים:

```
$ls
temp.c
term.c
```

הפקודת ls עם הדגל l- תציג את הפרטים הבאים:

```
$ls -l
-rw-rw-rw 1 andreas users 10553 Jan 9 08:54   temp.c
-rw-rw-rw 1 andreas users 520 Jan 10 12:23   term.c
```

3.3 פקודות עיבוד ברקע

לאחר שניתנה פקודה, יוצרת המעטפת תהליך חדש וממתינה עד אשר הוא יסיים את עבודתו. רק לאחר מכן היא מוכנה לקבל פקודה נוספת מהמשתמש.

אולם, פקודה המסתיימת ב-"&" תעביר את השליטה חזרה למעטפת בזמן שהיא מתבצעת, כך שהמעטפת תוכל לקבל את הפקודה הבאה. תפישה זו ידועה בשם עיבוד ברקע (background processing). לדוגמה, הפקודה

```
$cc -c test.c &
```

תבצע את ההידור (compile) של התכנית "test.c" ברקע, בעוד המשתמש יכול לבצע פקודות אחרות.

3.4 ביצוע של פקודות מקובצות

בין שתי הפקודות הנראות מטה יש הבדל קטן, שיש להבין אותו:

```
$cd /etc;cat passwd
$(cd /etc;cat passwd)
```

בפקודה הראשונה, כפי שתצפה בוודאי, המעטפת עוברת למדריך "/etc" ומציגה את תוכן קובץ הסיסמאות. לאחר ביצוע הפקודה, המעטפת נשארת ב-"/etc".

בפקודה השניה תופעל מעטפת חדשה. מעטפת זו תיעלם לאחר הביצוע של פקודת "cat" והמדריך של המעטפת שהפעילה אותה לא ישתנה.

3.5 תווים שמורים והצגת תמליל על המסך

במעטפת ישנם תווים מסוימים שיש להם משמעות כפולה, הן סמנטית והן תחבירית. תווים אלה, הידועים כתווים שמורים (metacharacters), מציינים פונקציות מיוחדות שיש לבצע בכל פעם שתו כזה מופיע. הרשימה המלאה של התווים השמורים, כמו '<', '>', '<<', '>>', נמצאת בנספח א'.

כאשר התו \ מקדים את התווים השמורים, הם מאבדים את משמעותם המיוחדת ומוצגים על המסך לפי צורתם הגרפית. לכן, כדי להציג על המסך את התו \ למשל צריך להקיש

```
$echo \\
```

ישנן שתי שיטות נוספות שבעזרתן ניתן להציג תווים: מרכאות רגילות ומרכאות כפולות.

כל התווים הנמצאים בין שתי סימני מרכאות רגילות (פרט לתווי המרכאות) יוצגו כפי שהם. כדי להציג את התמליל:

```
"*Error? Please reply (Y/N)"
```

השתמש בפקודה הבאה:

```
$echo '*Error? Please reply (Y/N)'
```

התווים הנמצאים בין שני סימני מרכאות כפולות יוצגו על המסך, פרט ל-", \, $, ו-', שאותם אפשר להציג בעזרת התו \ שיבוא לפניהם.

3.6 ניתוב קלט/פלט

במערכת ההפעלה UNIX, כל היחידות ההיקפיות (וגם מסוף המשתמש) נחשבים קבצים במערכת הקבצים. בצורה זו ניתן לבצע את כל הקלט והפלט על-ידי קריאה מקבצים או כתיבה אליהם.

כאשר המעטפת מבצעת פקודה, היא פותחת את מתארי (descriptors) הקובץ 0, 1 ו-2 הנקראים בהתאמה קלט סטנדרטי, פלט סטנדרטי ושגיאה סטנדרטית. מתארים אלה מתייחסים בדרך כלל למסוף של המשתמש וכך, ברירת המחדל של הפקודה היא לקרוא ממסוף המשתמש ולכתוב אליו. אולם לפני שפקודה מתבצעת, ניתן לנתב את הקלט או הפלט שלה בעזרת תווים שמורים. אם עומדים לבצע את התכנית "a.out" אז הפקודה

```
$a.out > out
```

תריץ את התכנית ותנתב את הפלט הסטנדרטי שלה לקובץ הנקרא "out".

הפקודה

```
$a.out < indata
```

תריץ את התכנית a.out עם נתונים שנקראו מהקובץ "indata".
ניתן לשלב את שתי הפקודות הקודמות בצורה הבאה:

```
$a.out < indata > out
```

קובץ הפלט "out" ייווצר בתנאי שהוא לא קיים, ואם הוא קיים - הוא יקוצץ לאפס. ניתן גם לצרף את התוצאות בסוף קובץ פלט קיים בעזרת התו השמור >> כפי שנראה להלן:

```
$a.out < indata >> out
```

נפוץ גם ניתוב בעזרת התחביר הבא:

ספרה&>	קישור של הקובץ לספרת מתאר הקובץ ובכך הופכת אותו לפלט הסטנדרטי.
ספרה&<	קישור של הקובץ לספרת מתאר הקובץ ובכך הופכת אותו לקלט הסטנדרטי.
<&-	סגירה של הקלט הסטנדרטי.
>&-	סגירה של הפלט הסטנדרטי.

ניתן להוסיף ספרה לפני כל אחת מצורות התחביר שנראו לעיל. לדוגמה, הפקודה

```
command 2 > out
```

תריץ את **command** ותנתב את פלט השגיאות הסטנדרטי לקובץ "out", בעוד שהפקודה

```
command 1>temp 2>&1
```

תריץ את **command** כאשר הפלט הסטנדרטי ופלט השגיאות הסטנדרטי מזווגים יחד; כלומר, היא מקשרת את מתאר הקובץ 2 עם אותו קובץ שאליו היה קשור מתאר הקובץ 1 (בדוגמה, הקובץ "temp").

בקבצי פקודות רבים במערכת אפשר לראות ניתוב ל-"/dev/null". זהו קובץ שאליו יוכלים להכנס כל המשתמשים והוא דומה ל"חור שחור", שתפקידו לבלוע פלט לא רצוי. לדוגמה, אם רוצים לעקוב אחר זמני הביצוע של התכנית "a.out", אשר מפיקה פלט להתקן הפלט הסטנדרטי (ברירת המחדל היא המסך של המשתמש), נוכל לנתב את הפלט של התכנית ל-"/dev/null" ואז יופיע על המסך המידע הקשור לזמן בלבד:

```
$time a.out >/dev/null
15.6 real 4.8 user 0.4 sys
```

3.7 שמות קבצים ותווים שמורים

כל פקודה הניתנת למעטפת, נבדקת לפני ביצועה, כדי לראות אם היא מכילה אחד או יותר מהתווים השמורים הבאים: *, ? ו-]. אם נמצא אחד מהתווים הללו, הפרמטר מוחלף בשמות הממוינים בצורה אלפאביתית, על פי התבנית הבאה:

*	מתאים לכל מחרוזת (כולל מחרוזת ריקה).
?	מתאים לכל תו בודד.
[...]	מתאים לכל אחד מהתווים הנמצאים בסוגריים. ניתן להשתמש בסימן -, כדי לציין תחום של תווים. הסימן ! לאחר הסוגר] מציין כל תו, פרט לאלה הנמצאים בתוך הסוגריים.

אם שמות הקבצים "test.a", "test.b", "test.c", "test.da" נמצאים במדריך הפעיל, אז השימוש בפקודת ls בצירוף התווים השמורים "?" ו-"*" יפיק את הפלט הנראה שלהלן:

```
$ls test.?
test.a
test.b
test.c
$ls test*
test.a
test.b
test.c
test.da
```

לדוגמה, נשתמש בפקודה

```
$ls [t-z]*
```

תפיק רשימה של כל הקבצים במדריך הנוכחי המתחילים באחת האותיות הנמצאות בין t ל-z כולל.

למעשה, זוהי אחריותה של המעטפת לבצע את ההרחבות ולהעביר את שמות הקבצים התואמים לפקודה. אם לא נמצאו שמות קבצים התואמים לתבנית מסוימת, תעביר המעטפת לפקודה את המחרוזת כפי שהיא, יחד עם התווים השמורים (כמו * ו-?).

3.8 צינורות ומסננים

ניתן להשתמש בקו האנכי '|', הידוע כסימן הצינור, כדי להריץ רצף של פקודות המהוות יחד צינור. במהלך ביצוע רצף הפקודות, הופך להיות הפלט הסטנדרטי של פקודה אחת לקלט הסטנדרטי של פקודה אחרת.

מסנן (filter) הוא פקודה, אשר קולטת מההתקן הסטנדרטי שלה, משנה את הקלט המגיע ומנתבת אותו לפלט הסטנדרטי שלה. לדוגמה, הפקודה

```
$who ¦ grep andy
```

תאתר ותדפיס את כל המשתמשים (אם ישנם כאלה) העובדים באותו רגע תחת השם "andy".

3.9 משתני המעטפת

ניתן לחלק את משתני המעטפת לשתי קטגוריות: פרמטרים המבוססים על מלות מפתח (keyword parameters) ופרמטרים מקומיים (positional parameters). בכל הקטגוריות, מציין התו $ את החלפת המשתנה.

פרמטרים המבוססים על מלות מפתח חייבים להתחיל באות ולאחריה אותיות נוספות, ספרות או קו תחתון. לדוגמה, ברמת הפקודה של המעטפת, פקודת ההשמה הבאה

```
$p=/work/acct/andy/sources
```

מקצה למשתנה "p" את המסלול המפורט בפקודה. ניתן לאחר מכן להשתמש בערך של "p" על ידי הקשה של הסימן $ לפניו. לדוגמה, הפקודה שלפנינו

```
$mv test $p
```

תעביר את הקובץ "test" מהמדריך הפעיל באותו רגע למדריך בשם /work/acct/andy/sources. אם נכתוב את הפקודה

```
$cd $p
```

היא תשנה את המדריך הנוכחי ל-"/work/acct/andy/sources".

אם מקישים לאחר הפרמטר אות, ספרה או קו תחתון שאין אנו רוצים שייחשבו כחלק משם המשתנה, עלינו להשתמש בסוגריים מסולסלים. לדוגמה,

```
$p=/work/acct/andy/sources/temp
$mv test ${p}2
```

פקודות אלו העבירו את הקובץ "test" למדריך

/work/acct/andy/sources/temp2

ישנם מספר משתני מעטפת המוגדרים מראש ומתוארים בקובץ "$HOME/.profile". קובץ זה יבוצע בזמן הכניסה למערכת ואז משתני המעטפת יקבלו את הערך שהוקצה להם. ניתן להשתמש

בפקודת **set** כדי לקבל רשימה של משתנים אלה (ראה פרק 4).

הבה נדון בנפרד בכל אחד מהמשתנים שבקובץ "$HOME/.profile".

- HOME - מכיל את שם מדריך הבית המשמש את פקודת **cd**. לדוגמה, כל אחת מהפקודות **$cd $HOME**, או **$cd**, תשנה את המדריך הנוכחי הפעיל למדריך הבית.
- IFS - מציין את מפרידי השדה הפנימי (internal field seperators) שמחלקים את הפקודה. ערכי ברירת המחדל הם בדרך כלל **space**, **tab** ו-**newline**.
- MAIL - מציין את שם הקובץ שהמעטפת בודקת, כדי למצוא אם המשתמש קיבל הודעות חדשות.
- MAILCHECK - מציין את התדירות (בשניות) שבה המעטפת צריכה לבדוק אם התקבלו הודעות חדשות.
- PATH - מציין את מסלול החיפוש של פקודות. לדוגמה, הפקודה PATH=/bin:/usr/bin תחפש פקודה מסוימת במדריך /bin תחילה ולאחר מכן במדריך "/usr/bin".
- PS1 - מציין את המחרוזת הראשית של המנחה (prompt). בזמן שנכנסים למערכת, המחרוזת הראשית היא, בדרך כלל, סימן $, או שם המערכת.
- PS2 - מציין את המחרוזת המשנית של המנחה, שתופיע אם הפקודה תהיה מתכולה של שורה אחת.
- SHELL - מציין את סוג המעטפת שבשימוש. לדוגמה, "/bin/sh" למעטפת בורן, "/bin/csh" למעטפת C ו-"/bin/rsh" למעטפת מוגבלת.
- TERM - מציין את סוג המסוף שמשתמשים בו (לדוגמה, vt200).
- TERMINFO - אפשרויות ותכונות המסוף מתוארות בבסיס נתונים הנמצא ב-"/usr/lib/terminfo". כדי לבחון בקלות תיאור של מסוף חדש, מציבים במשתנה TERMINFO את שם המסלול, אשר כולל את התיאור של TERMINFO.
- TERMCAP - דומה למשתנה TERMINFO. משתנה זה מתייחס לאפשרויות המסוף המוגדרות בבסיס הנתונים "/etc/termcap".

כאשר מתבצע קובץ פקודות של המעטפת, ניתן להכנס לארגומנטים שלו בעזרת הפרמטרים המקומיים $0, $1, ...$9. הפרמטר $0 מכיל תמיד את שם קובץ הפקודות שמתבצע. לדוגמה, כאשר מפעילים קובץ פקודות של המעטפת הנקרא "weekday" בצורה הבאה

```
$weekday 6 7
```

משמעות הפרמטרים הינה כדהלן:

$0	יכיל את הערך **weekday**.
$1	יכיל את הערך **6**.
$2	יכיל את הערך **7**.
$3-$9	לא יוגדרו.

למספר פרמטרים נוספים אשר הוגדרו, ניתן לפנות בתוך קובץ הפקודות:

$# מספר הפרמטרים המקומיים. בדוגמה "weekday", ערך הפרמטר $# יהיה 2 מכיון ששני פרמטרים הועברו לפקודה.
$! מספר התהליך של הפקודה האחרונה שרצה ברקע.
$$ מספר התהליך של מעטפת זו. כדאי להשתמש בפרמטר זה ליצירת קובצי עבודה זמניים ויחודיים (כמו למשל ls>temp$$).
$- דגלי המעטפת הנוכחיים, אשר סופקו למעטפת בעת שהופעלה, או באמצעות הפקודה **set**.
$? המצב שבו הסתיימה הפקודה האחרונה שבוצעה. רוב הפקודות מחזירות ערך עשרוני השווה לאפס אם הן מסתיימות בהצלחה. על כן ניתן להשתמש בפרמטר זה כדי לבדוק הצלחה או כישלון של פקודה.
$* פרמטר זה יתאים לכל הפרמטרים שסופקו, להוציא פרמטר $0.

בסעיף 3.15 ניתן לראות קובצי פקודות של המעטפת המשתמשים בפרמטרים שהוסברו לעיל.

3.10 הצבה בפקודות

הצבה בפקודות הינה אחד המאפיינים החזקים ביותר של המעטפת. את הפלט הסטנדרטי של פקודה, הנמצא בין מרכאות רגילות ('), ניתן להציב לתוך משתנה, שגיאה נפוצה היא להשתמש במרכאות כפולות במקום מרכאות רגילות. שים לב לכך.

הפקודה **tty** תזהה את המסוף הפעיל באותו רגע ותציין אותו בצורה "/dev/tty<n>", כאשר <n> הינו מספר עשרוני. כדי להדפיס את הפלט "/dev/tty<n>" נכתוב את הפקודות הבאות:

```
$mytty='tty'
$echo $mytty
```

אם יש צורך להדפיס רק tty<n> מתוך /dev/tty<n>, ניתן להשתמש בפקודת **set** ובפקודת **who am I** כדי לעשות זאת. לדוגמה, נשתמש בפקודה

```
who am I
```

ונקבל את הפלט הבא:

```
andreas tty01 Jan 22 10:30
```

עתה, כדי לקבל את שם המסוף (כלומר, את השדה tty01), נשתמש בפקודות ההצבה הבאות:

```
$set 'who am I'
$mytty=$2
$echo $mytty
```

השדה "tty01" יוצג במסוף, מכיון שהפרמטרים $1, $2, $3, $4 ו-$5 יהיו בהתאמה "andreas", "tty01", "Jan", "22" ו-"10:30", ואנו מבקשים את פרמטר 2 בלבד.

3.11 פקודות שימושיות

נציג עתה מספר פקודות נפוצות בקבצי פקודות של מעטפת. מספר קבצי פקודות המדגימים איך להשתמש בפקודות אלו נמצאים בחלק האחרון של פרק זה.

- **echo** - מציגה את הארגומנטים שלה על התקן הפלט הסטנדרטי (לדוגמה, echo $TERM תציג את הערך של המשתנה TERM).

- **read** - משמשת לקריאת תווים מערוץ הקלט הסטנדרטי. לדוגמה, כדי לקרוא את התאריך שהמשתמש סיפק ולהציג אותו שוב, השתמש בפקודות הבאות:

```
$echo "Please enter date: \c"
$read date
$echo $date
```

- **test (ביטוי)** - הפקודה **test** מעריכה את ה"ביטוי" ואם הערך שלו הוא נכון, היא מחזירה אפס (אמת). במקרים אחרים היא מחזירה ערך שאינו אפס (שקר). הביטוי יכול להיות מורכב מהמשפטים הבאים:

שם-קובץ	-r	אמת, אם הקובץ ניתן לקריאה (למשל, test -r abc.c).
שם-קובץ	-w	אמת, אם הקובץ ניתן לכתיבה
שם-קובץ	-x	אמת, אם הקובץ ניתן לביצוע
שם-קובץ	-s	אמת, אם נפח הקובץ גדול מאפס
שם-קובץ	-u	אמת, אם נקבעה סיבית לזיהוי המשתמש
שם-קובץ	-d	אמת, אם הקובץ הוא מדריך
el op e2		אשר op יכול להיות -ne, -eq, -gt, -ge, -lt ו-le-. משמעותן של אופציות אלו הוא בהתאמה: לא שווה, שווה, גדול מ-, גדול מ- או שווה, קטן מ-, וקטן מ- או שווה.

- **expr** - משמשת להערכת ארגומנטים. לדוגמה, אם נכתוב את ההוראות הבאות, הן ידפיסו את הערך 34.

```
$a='expr 2 + 4 \* 8'
$echo $a
```

שים לב, שיש להקיש תו Escape (לוכסן הפוך) לפני הסימן '*', שאם לא כן, נקבל רשימה של הקבצים במדריך שפעיל באותו רגע.

- **[תהליך] wait** - ממתינה עד אשר התהליך המצוין בסוגריים ישלים את עבודתו וידווח על מצבו בעת סיום העבודה. אם לא צויין תהליך, תהיה ההמתנה עד אשר כל תהליכי הבנים הפועלים באותו רגע יחזירו קוד אפס.

- **sleep** - משמשת להשעיית הביצוע של התהליך למספר שניות הנקוב בפקודה. לדוגמה,

```
(sleep 10; ls -l ¦ more)
```

תהיה המתנה של 10 שניות ואח"כ תבוצע הפקודה שנמצאת אחרי הסימן נקודה-פסיק.

- **wall** - שולחת הודעות לכל המשתמשים במערכת. לדוגמה,

```
$echo "System going down in 60 seconds" ¦ /etc/wall
```

3.12 אותות ומלכודות

אותות (signals) יכולים להיות מופקים כתוצאה מהתרחשות אירועים חריגים כמו פקודת kill, או טעות בתכנית, שגיאת אפיק (bus error), או הקשה על DEL. במקרים רבים, אות יגרום להפסקת התהליך המקבל, אבל במקרים רבים אחרים ניתן ללכוד את האות ולהתעלם ממנו, או להשתמש בו כדי להעביר את השליטה למקום כלשהו אחר בתכנית או במערכת ההפעלה. בטבלה 3.1 ניתן למצוא רשימה של האותות הנפוצים ב-UNIX.

פקודת המלכודת **trap** מיועדת ללכוד את האותות הנכנסים:

```
trap <רצף-פקודות> <מספרי-האותות>
```

לדוגמה, אם נרשום את הפקודה

```
trap "" 1 2 3
```

היא תחסום את פסקי המקלדת (כיון שבהתקבל אחד האותות הללו תתבצע הפקודה "", שהיא פקודה ריקה). פירושו של דבר, שהמשתמש לא יוכל לעצור את ביצוע קובץ הפקודות על ידי האותות SIGHUP, SIGINT, ו-SIGQUIT. כדי להחזיר את פסקי המקלדת יש להשתמש בפקודה הבאה:

```
trap 1 2 3
```

כדי לבטל את כל קובצי "*temp" שבמדריך "/usr/test" ולהפסיק את ביצוע קובץ הפקודות (באמצעות פקודת exit), לאחר שמתקבלים אותות 2 או 3, אפשר להשתמש בפקודה הבאה:

```
trap "rm /usr/test/temp*;exit" 2 3
```

טבלה 3.1 אותות ה-UNIX

SIGHUP	1	ניתוק
SIGINT	2	פסק
SIGQUIT	3	יציאה
SIGILL	4	הוראה לא חוקית (לא אותחלה בזמן שנתפסה)
SIGTRAP	5	מלכודת לא חוקית (לא הותחלה בזמן שנתפסה)
SIGIOT	6	הוראת TRAPV
SIGEMT	7	הוראת EMT
SIGFPE	8	חריגה בנקודה צפה
SIGKILL	9	ביטול (לא ניתן ללכוד או להתעלם)
SIGBUS	10	שגיאת אפיק
SIGSEGV	11	הפרה של כללי החלוקה לסגמנטים
SIGSYS	12	ארגומנט לא תקין בפקודת קריאה של המערכת
SIGPIPE	13	כתיבה לצינור, שאין מי שיקרא ממנו
SIGALARM	14	שעון מעורר
SIGTERM	15	אות סיום/הפסקה של תכנית
SIGUSR1	16	אות 1 המוגדר על ידי המשתמש
SIGUSR2	17	אות 2 המוגדר על ידי המשתמש
SIGCLD	18	חיסול של ילד
SIGPWR	19	תקלה במתח החשמל או שיתחול (restart)

ניתן להשתמש בפקודה **kill** כדי לשלוח אות לתהליך. לדוגמה, את האות המופק על-ידי הפקודה

```
$kill -15 <מס' מזהה של התהליך>
```

ניתן ללכוד בעזרת הפקודה **trap**. אולם הפקודה

```
$kill -9 <מס' מזהה של התהליך>
```

תמיד תפסיק את פעולתו של התהליך, מכיון שלא ניתן ללכוד את SIGKILL (אות מס' 9), או להתעלם ממנו. חשוב לזכור, שאם אות אינו מפסיק את פעולתו של תהליך שרץ, התהליך ימשיך את פעולתו, אלא אם צוינה במפורש הפקודה **exit**.

3.13 שימוש במסמכי Here

ניתן להשתמש במסמך Here כדי לבצע רצף של פקודות. רצף הפקודות מתחיל בתווים '<<' ולאחריהם רושמים סימן כמו '%', או מחרוזת כמו 'END'; גוף המחרוזת מסתיים בפעם הבאה שבה מופיע הסימן, או המחרוזת שרשמנו לאחר התווים '<<'. על ידי ציון הסימן שלאחר '<<', ניתן למנוע החלפה של הסימנים $, \ ו-'.

הבה נתבונן במקרה שבו יש להמיר את כל האותיות הקטנות בקובץ באותיות גדולות. ניתן להשתמש לשם כך בפקודה **tr** (translate), או במסמך Here.

אפשר להשתמש בפקודה **tr** בצורה הבאה:

```
$cat infile ¦ tr "[a-z]" "[A-Z]" > outfile
```

"infile" הינו קובץ הקלט ו-"outfile" הינו קובץ הפלט.

אפשר לבצע את המרת האותיות הקטנות בגדולות באמצעות מסמך Were. לצורך זה יש לשלב בקובץ הפקודות את תכנית העריכה **ed**, ואת הפקודות שלה, כפי שנראה להלן:

```
ed $1 <<%
g/a/s//A/g
g/b/s//B/g
,
,
,
g/y/s//Y/g
g/z/s//Z/g
w
q
%
```

אם קובץ הפקודות הנ"ל ייקרא "upper", נוכל להשתמש בפקודה

```
$upper infile
```

כדי להמיר את כל התווים בקובץ "infile" לאותיות גדולות.

3.14 תבניות תכנות במעטפת

בקבצי פקודות של המעטפת ניתן לכלול פקודות המשתמשות במפעילים (operators) לוגיים. לדוגמה, ניתן להשתמש במפעיל = (שווה) ובמפעיל-! (לא שווה) כדי להשוות מחרוזות. ניתן להשתמש במפעיל && ("andif") ובמפעיל || ("orif") כדי לבצע את הפעולות הלוגיות AND ו-OR בהתאמה.

לדוגמה, המשפט

```
test $uid -le 100 && uid=60
```

יבדוק את ערך המשתנה "uid" ואם הערך שלו קטן או שווה ל-100, הוא יקבל את הערך 60. אם נשתמש במפעיל "||" במקום במפעיל "&&", נקבל פעולה הפוכה. כלומר, באמצעות המשפט הבא

```
test $uid -le 100 || uid=60
```

ניתן למשתנה "uid" את הערך 60, רק אם ערכו קטן או שווה ל-100.

בנוסף למפעילים אלה הוגדרו מספר מלים שמורות שניתן להשתמש בהם ליצירת מבני תכנות חזקים ביותר. להלן רשימת המלים השמורות:

if	then	elif	else	fi	case	
for	in	while	until	do	done	esac

בסעיפי המשנה שלהלן נסביר את המבנים של הפקודות הבאות: **if, case, while, until ו-for.**

המבנים if ו-case מאפשרים למשתמש להתנות ביצוע של פקודות. המבנים while, until ו-for מאפשרים ליישם לולאות, אשר בהן סדרת פקודות מתבצעת עד אשר תנאי, או מספר תנאים מתקיימים.

תבנית התכנות if

המבנה של משפט if:
(הסוגריים משמשים לתיחום מלת ההסבר ואינם חלק מן הפקודה)

```
if   (תנאי)
     then
     (משפט/ים)
elif  (תנאי)
     then
     (משפט/ים)
 else
 (משפט/ים)
 fi
```

הקטעים המכילים את **elif** ו-**else** הינם רשות. לדוגמה, כדי לבדוק שקובץ קיים וניתן לביצוע נכתוב:

```
if test -x swap.c
then
    echo "File swap.c was found\n"
else
    echo "File swap.c was not found\n"
fi
```

תבנית התכנות case

כל אשר ניתן להשיג באמצעות מספר משפטי **if** ו-**elif** ניתן לכתוב במשפט **case** אחד, אשר מבוסס על התאמת תבניות. במשפט **case** ניתן להשתמש בתו השמור **'*'**, המציין מספר כלשהו של תווים ובתו **'?'**- המתאים לכל תו בודד. ניתן להשתמש גם בסוגריים המרובעים כדי לציין תחום. משפט **case** הוא בעל המבנה הבא (הסוגר הימני וסימני ; הם חלק מן הפקודה):

```
case מלה in
    1אופציה  )  משפט/ים;
    2אופציה  )  משפט/ים;
          .
          .
          .
    yאופציה  )  משפט/ים;
 esac
```

בדוגמה שלהלן, השתמשנו במשפט **case** לזיהוי מערכת בעזרת הארגומנט שהמשתמש מעביר לקובץ הפקודות שהפעיל אותה. התו '*' הינו ברירת המחדל, שתפקידו לציין תו כלשהו:

```
name=$1
'ase $name in
     20) echo "Its system 20\n";;
     40 | 80 ) echo "Its system 40 or 80\n";;
     * ) echo "System can not be identified\n";;
```

לולאת while

לולאת **while** הינה בעלת המבנה הבא (הסוגריים אינם חלק מן הפקודה):

```
while  (תנאי)
  do
   (משפט/ים)
  done
```

קובץ הפקודות הבא יכול לפעול ברקע ולהדפיס הודעה כאשר משתמש בשם 'andy' נכנס למערכת:

```
while ( who | grep andy > temp )
do
  sleep 15
done
echo  "User 'andy' has just logged in on the system\n"
```

לולאת until

לולאת **until** הינה בעלת המבנה הבא (הסוגריים אינם חלק מן הפקודה):

```
until  (תנאי)
  do
    (משפט/ים)
  done
```

לדוגמה, כדי לבדוק אם תהליך שרץ יצר קובץ בשם 'data',

ואם לא - לבצע בדיקה חוזרת בכל 10 שניות, יש לכתוב את הפקודות הבאות:

```
until test -f data
do
  sleep 10
done
echo "File data has now been created\n"
```

לולאת for

לולאת **for** הינה בעלת המבנה הבא:

```
for שם [in רשימת-שמות]
 do
   משפט/ים
 done
```

בדוגמה הבאה, לולאת **for** משמשת לחישוב והדפסת הערך של ריבוע המספרים 1, 2, 3 ו-4:

```
for i in 1 2 3 4
do
  a='expr $i \* $i'
  echo "Square of $i=$a\n"
done
```

3.15 סיכום - דוגמאות של קובצי פקודות

ניתן ליצור קובץ פקודות של המעטפת בעזרת תכנית עריכה, כמו **vi** למשל. שורה בקובץ הפקודות המתחילה בסימן "#" מתורגמת כשורת הערה שאינה מבוצעת. לאחר שנוצר קובץ הפקודות ונשמר, יש לשנות את אופן (mode) ההפעלה שלו, כדי שניתן יהיה לבצע אותו. אפשר לעשות זאת על ידי שימוש בפקודה

```
$chmod  +x  שם-קובץ-הפקודות
```

או בפקודה

```
$chmod  0777  שם-קובץ-הפקודות
```

שתי הפקודות גורמות לכך שאתה, הקבוצה שלך וכל שאר המשתמשים תוכלו לבצע את קובץ הפקודות (ראה פקודת **chmod**). נסכם את הפרק בהצגה של מספר קבצי פקודות המדגימים את אשר למדנו.

חסימת פסקים מהמקלדת

בקובץ הפקודות הבא ישנה לולאה שתפעל עד אשר תקיש "y", או עד שתשתמש בפקודה **kill 9 <proces-id>**. לא ניתן להפסיק פקודה זו באמצע, כמו למשל בהקשת DEL, מכיון שהשתמשנו בפקודה **trap** כדי לבטל אפשרות של קבלת אות מספר **2**. שים לב לשימוש בתבנית התכנות **if** עם סוגריים מרובעים:

```
# Script  : loop
#

trap "" 1 2 3
# the while construct below can also be written as: while true
while :
     do
       #the "\c" character below is used to suppress the newline
       echo "Enter any string or y to end: \c"
       read a
       # if construct below is the same as: if test "$1" = "y"
       if [ "$a" = "y" ]
         then
            exit
       fi
     done
```

מספור שורות בקובץ והדפסתו

קובץ הפקודות הבא משתמש בפקודת **awk** כדי למספר שורות בקובץ. לאחר מכן הוא מדפיס את הקובץ בתנאי ששני ארגומנטים כלשהם הועברו לקובץ הפקודות. אם לא כן, הוא מציג את הקובץ, מסך אחר מסך, באמצעות הפקודה **more**.

```
#Script: lplines
#
 if [ $# -eq 0 ]
  then
```

```
      echo "Please specify file name to be printed \
            with line numbers\n"
      echo "Usage is 'lplines filename lp'"
      echo "If 'lp' is missing 'more' is assumed\n"
      exit 1
fi
# note that only the number of arguments are checked for.
# no attempt is made to check for value lp
if[ $# -ne 2 ]
  then
      awk '{printf "%3d %s\n", NR, $0}' $1 | more
else
      awk '{printf "%3d %s\n", NR, $0}' $1 | pr -e | lp "-t$1"
fi
```

כדי למספר שורות של קובץ הנקרא "test.c" ולהדפיס אותו במדפסת, צריך להקיש את הפקודה הבאה:

```
$lplines test.c lp
```

הידור והעברת שגיאות תחביריות לקובץ

הפקודה **cc** משמשת להידור תכניות הכתובות בשפת C. בקובץ הפקודות שלהלן, הפלט הסטנדרטי מקושר עם הקובץ "temp" שאליו יועברו שגיאות תחביריות (אם ישנן כאלה).
קובץ השגיאות הסטנדרטי משולב בקובץ הפלט הסטנדרטי:

```
#Script : compdirect
#
echo "Compiling $1 and directing syntax errors to file temp\n"
cc $1 1>temp 2>&1
```

כדי להדר קובץ הנקרא "swap.c" צריך להקיש את הפקודה:

```
$compdirect swap.c
```

הדפסת כל הקבצים שבמדריך

לפעמים יש צורך להדפיס את כל הקבצים שבמדריך. קובץ הפקודות הבא ימספר וידפיס את כל המדריך ששמו הועבר כארגומנט.

```
#Script : alllines
#
 for i in $*
```

```
do
   echo "================$1====================\n"
   echo "=======================================\n"
   cat $i | awk '{printf "%3d %s\n",NR,$0}'
done | pr -e -172 | lp
```

כדי להדפיס את הקבצים במדריך "/usr/include", צריך להקיש את הפקודה

```
$alllines /usr/include
```

חישובים אריתמטיים

קובץ הפקודות "calc" שלהלן מדגים את השימוש בפקודת **expr** לביצוע חישובים ולמציאת אורך של מחרוזת.

```
# Script : calc
#
# Note    : Brackets and "*" character must be escaped by a
#           backslash
        # Use expr for addition and division
        a='expr \( 350000 + 77000 \) / 67'
        echo "a=$a \n"

        # Use expr for multiplication
        d='expr $a \* 2'
        # echo "d=$d \n"
# Also use expr to count number of characters in a variable
# e.g. "Andreas" is seven characters long, so it will print
# the number 7 in the example below
name="Andreas"
expr $name : '.*'
```

בניית תפריט

ניתן להשתמש בקובץ פקודות לבניית תפריט פשוט כפי שנראה להלן. משפט "case" משמש לזיהוי הקלט של המשתמש. קובץ הפקודות יסתיים כאשר המשתמש יקיש את הספרה 3.

```
# Script : genmenu
#
```

```
trap "" 2
a=0     # initialize variable "a" to zero
while true    # infinite loop
do
 clear      # clear the screen
 echo "\n\n\n\n\n"
 echo "                    M E N U  - SYSTEM FUNCTIONS\n"
 echo "                   ===========================\n\n"
  echo "                    1 = Print the date \n\n"
 echo "                   2 = Print who is logged on the system\n\n"
 echo "                   3 = Exit\n\n\n"
 echo "                   Please enter selection : \c"
 read a
 clear
 case $a in
        1) date;;
        2) who;;
        3) clear;exit;;
        *) echo "$a = Invalid option number !!";;
 esac
 echo "\nHit return to continue : \c"
 read b
done
```

עריכת קבצים עם מסמך Here

קובץ הפקודות "allfor" הנראה להלן מראה כיצד ניתן לשנות במדריך מסוים את כל הקבצים עם הסיומת ".f". הפעולה נעשית בעזרת תכנית העריכה **ed** במסמך Here.

```
# Script : allfor
#

clear
for file in *.f
 do
  echo "\n$file"
  ed - $file <<!          # start of here document
  1a                      # go to line 1 in the file and append lines
  \$CHAREQU               # puts on first line $CHAREQU
  \$INT2                  # puts on second line $INT2
                          # fullstop terminates the "ed" input mode
```

```
 # now change all lines in the file that contain the
 # value "LOGICAL " to "LOGICAL*2 "
 g/LOGICAL /s//LOGICAL*2 /g
 w                    # save the changes that were made
 q                    # quit the ed editor
 !                    # end of Here document
done                  # end of for loop
```

קריאות לפונקציות בקובצי פקודות

קובץ הפקודות שלהלן מראה כיצד לכתוב ולהפעיל פונקציות בקובצי פקודות. הפונקציה **fnd** מיועדת להציג את מספר השורה בקובץ "test", המכיל את המלה "nancy".

```
# Script : locate
#

fnd()
{
no=0
cat test | while read line
     do
     # note the use of semicolon before 'then'
     if [ "$line" = "nancy" ] ; then
        # echo line number on which the word 'nancy' was found
        echo $no
     fi
     # increase line number counter
     no='expr $no + 1'
     done
}

  # invoke function 'fnd' and assign counter that it displays
  # to the variable 'found'
  # note that accent graves are used in the function call below
  found='fnd'
  if [ $found != 0 ] ; then
     echo "$0:Found the string 'nancy' on line $found\n"
  else
     echo "$0:Can not locate the string 'nancy'\n"
  fi
```

פרק 4

שינויים בסביבת המעטפת

על משתמשי UNIX להבין את סביבת המעטפת כדי שיוכלו להשתמש במערכת ההפעלה לפיתוח יישומים. פרק זה מסביר, בעזרת מספר דוגמאות, כיצד לטפל בסביבת המעטפת ולשנותה מרמת הפקודה של המעטפת, מתוך קובצי פקודות (scriptfiles), ומתוך תכניות הכתובות בשפת C.

4.1 משתני המעטפת

הסביבה היא רשימה המכילה זוגות של משתנה-ערך בצורה הבאה:

ערך=משתנה vaiable=value

את הרשימה ההתחלתית של הזוגות משתנה-ערך, המשמשת לקביעת סביבת המשתמש, ניתן למצוא בקובץ "/etc/profile" (במעטפת Bourne). קובץ זה יבוצע בזמן הכניסה למערכת ואז יוקצה לכל ערך מתאים.

ניתן לקבל רשימה של משתני הסביבה הנראים מטה, בעזרת הפקודה "env" (ראה הסבר אודות המשתנים בפרק 3).

```
$env
HOME      = /work/acct/andy
IFS       =
MAIL      = /usr/mail/andy
MAILCHECK= 600
PATH      = /bin:/usr/bin:/etc::
PS1       = $
PS2       = >
SHELL     = /bin/sh
TERM      = svt120
```

ניתן ליצור זוגות חדשים של משתנה-ערך וניתן לשנות את הקיימים בעזרת הצבות פשוטות ופקודת **export** כפי שנראה בהמשך.

לדוגמה, כאשר נכתוב את הפקודות

```
$TERM=ampex
$export TERM
```

ישתנה הערך של משתנה הסביבה TERM ל-"ampex". באופן דומה, כאשר נכתוב את הפקודות

```
$mydir=/usr/andy/temp
$export mydir
```

הן יכניסו את המשתנה "mydir", המוגדר על-ידי המשתמש, לסביבת העבודה. בצורה זו ניתן להשתמש בפקודת **export** כדי לקשור משתנים חדשים לסביבה.

דרך אחרת לקשירת זוגות של משתנה-ערך לסביבה, היא להשתמש באמצעות הפקודה **set -a**. לדוגמה:

```
$set -a
$TERM=wyse
$objs=/usr/andy/objects
```

במקרה זה תכניס הפקודה **set -a** לסביבה את כל הזוגות שבאים אחריה. כדי לבטל את ההשפעה של **set -a** השתמש ב-**set +a**.

ניתן להשתמש בפקודת **unset** כדי לבטל זוגות של משתנה-ערך. לדוגמה, אם נכתוב

```
$unset TERM
```

פקודה זו תבטל את המשתנה TERM מתוך הסביבה. שים לב, לא ניתן לבטל משתנים אלה: IFS, PS2, PS1, PATH, MAILCHECK.

תרגיל מעניין בשלב זה יהיה להשתמש בפקודות **set** ו-**env**, לאחר שנשתמש בפקודות **export**, **set -a** ו-**unset**, כך ניתן להבין את ההבדל בין **set** לבין-**env**.

4.2 הסביבה של תכנית C

רשימה של זוגות משתנה-ערך, הקובעים את הסביבה, מועברת לתכנית המתבצעת בדרך שמועברת רשימת ארגומנטים. כאשר תכנית C מתבצעת, הפקודות שמפעילות אותה נראות כך:

```
main(argc,argv,envp)
int argc;
char *argv[], *envp[];
```

"argc" הינו מונה הארגומנטים. "argv" הינו מערך של מצביעי

תווים (character pointers) למחרוזות המסתיימות בתו ריק (null-terminated string). "envp" הינו מערך של מצביעי תווים למחרוזות הקובעות את הסביבה של התכנית. ניתן לציין במפורש בעזרת פקודת **execle** את הסביבה שבה תכנית מסוימת תתבצע כפי שנראה להלן:

```
char  *envp[]={
            "HOME=/usr/andy",
            "TERM=wyse",
            .
            .
            .
            0 /* list must be terminated by zero */
            };
main()
{
execle("/bin/sh","sh",envp);
.
.
}
```

שים לב שבדוגמה לעיל השינויים הם בסביבה שבה התכנית מתבצעת בלבד, ולא בסביבת האב שמתוכה הופעלה התוכנית (סביבת האב יכולה להיות, לדוגמה, הסביבה של קובץ פקודות של המעטפת שמתוכו הופעלה התכנית). בדיון שלהלן נראה כיצד אפשר לשנות גם את סביבת האב.

ניתן לקבל את פירוט הסביבה של תכנית מסוימת על ידי שימוש במשתנה החיצוני "environ" כפי שנראה להלן.

```
extern char **environ;
char **penv;
main()
{
int d=1;
penv=environ;
while(*penv)
      {
      printf("Environment variable-%d=%s\n",d,*penv++);
      d++;
      }
}
```

ניתן להשתמש בשגרת הספריה "getenv" כדי לבחון את משתני הסביבה ובשגרה "putenv" - כדי לשנות את ערך משתנה הסביבה הקיים או כדי להוסיף לסביבה זוג חדש של משתנה-ערך. השגרה "putenv"- משיגה את השטח הדרוש למשתנים החדשים על ידי קריאה לשגרת המערכת "malloc".

התכנית בשפת C שלהלן מדגימה את השימוש בשתי השגרות.

```
char keep[40];
main()
{
char y[40];
if( m_penv ("TERM=svt120") )
     {
     printf("Error: cannot modify environment\n");
     exit(1);
     }
if  m_genv("TERM", y) )
     {
     printf("Error: cannot read environment\n");
     exit(2);
     }
else
     printf("New terminal name=%s\n",y);
exit(0);
} /* end of main program */
/* Routine       : To change value of an environment
                   variable */
static int m_penv(s)
char *s;
{
extern char *putenv();
int flag=0;
strcpy(keep,s);
if( putenv(keep) )
     flag = 1; /*error occurred*/
return(flag);
}

/* Routine       : To read value of an environment
                   variable */
static int m_genv(s,y)
char *s, *y;
{
```

```
char *a;
extern char *getenv();
int flag=1;
a=getenv(s); /* read environment variable */
if(a != (char *)0)
     {
     /* the environment variable was read OK */
     strcpy(y,a);
     flag=0;
     }
return(flag);
}
```

4.3 שינויים בסביבת מעטפת האב

שגרת "putenv" מטפלת בסביבה שצוינה על ידי המשתנה "environ" (שהוסבר קודם) אך אינה משנה את "envp", הארגומנט השלישי של "main". השאלה הבאה מתעוררת בשלב זה היא, כיצד ניתן לשנות את "envp", כך שכל שינוי שנעשה בסביבה על ידי התכנית שנקראה מקובץ הפקודות יהיה זמין גם לתכניות אחרות באותו קובץ פקודות?

כאשר תכנית מתבצעת, היא יוצרת מעטפת חדשה וכל שינוי בזוגות של משתנה-ערך ישפיע רק על המעטפת החדשה ולא על מעטפת האב. כדי לעקוף בעיה זו, ניתן להכניס את הזוגות משתנה-ערך לקובץ זמני מיוחד ולהשתמש בפקודה

. file name

שבה רווח מפריד בין נקודה (full stop) לשם-קובץ. קובץ פקודות טיפוסי ייראה כך:

```
.
.
set 'who am I'    # execute 'who am I' command
ttyname=$2        # get terminal name
prog1             # execute C program 'prog1'
if [ -f $ttyname ]
  then  # modify environment with variables in ttyname
     . $ttyname
     rm $ttyname # remove file name
fi
prog2             # execute C program 'prog2'
.
.
```

בדוגמה לעיל, בהנחה ש-prog1 צריכה לשנות את המשתנים TERM ו-PATH השייכים לסביבת קובץ הפקודות, היא יכולה ליצור קובץ המכיל זוגות של משתנה-ערך בדומה לזה שלהלן:

```
TERM=svt120
export TERM
PATH=/work/acct/proj/files
export PATH
```

משפט **if** בקובץ הפקודות של המעטפת בודק אם קובץ כזה קיים. אם הקובץ קיים, הוא משתמש בפקודה **fullstop** כדי לשנות את סביבת האב של קובץ פקודות המעטפת. שינויי הסביבה שנעשו על ידי "prog1" יהיו זמינים גם ל-"prog2" (שים לב שניתן להשתמש בפקודת **fullstop** בדרך דומה, כדי לשנות את הסביבה ברמת פקודות המעטפת).



פרק 5

הפילטרים ב-UNIX ותכנית awk

מערכת UNIX כוללת מספר רב למדי של פקודות ותכניות שירות שמאפשרות למשתמש לעבד קבצים ופלט של פקודות בקלות רבה וביעילות.

פרק זה יתאר רבות מפקודות אלה, וביניהן את תכנית השירות **awk** הנחשבת לשפת התכנות המורכבת ביותר המשמשת להתאמת תבניות (pattern-matching) ולעיבוד תמלילים.

בהמשך ניתן סקירה על הפקודות הבאות:

grep	**wc**	**cmp**
split	**cut**	**tr**
paste	**od**	**sort**

5.1 תכנית השירות awk

awk הינה תכנית שירות להתאמת תבניות וגם שפה המושפעת מאד משפת C. בנוסף לכך שהיא משתמשת ברעיונות משפת C, היא משתמשת גם ברעיונות מתכנית השירות **sed** של UNIX ומשפת SNOBOL. **awk** נקראת על שם שלושת מפתחיה: Al Aho, Pet Weinberger, Brian Kerningham. ניתן להשתמש ב-awk למשימות כמו אלה הרשומות להלן:

* המרה של נתוני קלט למבנה שיתאים לעיבוד על ידי תכנית אחרת.
* עריכת דוחות.

הפעולות הבסיסיות שתכנית awk מבצעת:

* קריאת רשומה.
* הפרדת רשומה לשדות.
* ביצוע הפעולה המבוקשת.
* חזרה על שלבים אלה עד סוף הקובץ.

כל רשומת קלט מחולקת לשדות המופרדים על ידי התווים 'tab'

(טאבולטור), או 'space' (רווח). אפשר לגשת לכל שדה בעזרת המשתנים השמורים $1, $2 וכו', כאשר הסימון $1 מתייחס לשדה הראשון, $2 מתייחס לשדה השני וכן הלאה. המשתנה השמור $0 מתייחס לשורת קלט, או לרשומה שלמה. גם המלים BEGIN ו-END הינן מלים שמורות, אשר משמשות בדרך כלל לציון תחילה וסוף של תכנית.

ניתן להשתמש במספר פעלים לוגיים (אופרטורים) בשפת **awk**. פעלים אלה מפורטים בטבלה 5.1 בסדר חשיבות יורד.

טבלה 5.1 פעלים לוגיים בשפת awk

פעלים לוגיים	מטרה (קרא מימין לשמאל בהתאמה)
++, --	הוספה, גריעה
/,*,%	חילוק, כפל, שארית
+,-	חיבור, חיסור
=!,<,>,=<,=>,==	סימן אי-שיוויון (=!) וסימני יחס אחרים.
!	הפיכת הערך של ביטוי.
&&	הפעולה הלוגית AND (לדוגמה, a&&b)
||	הפעולה הלוגית OR (לדוגמה, a||b)
=,=*,=-,=+,=/,=%	זהה לפעולות בשפת C.

טבלה 5.2 מציגה כמה מאפייני **awk** שיכולים לשמש בייצוג מספרי.

טבלה 5.2 סוגים שונים של ייצוג מספרי

הסוג	המרה ל
%d	מספר עשרוני
%f	נקודה צפה
%o	אוקטלי
%x	הקסדצימלי
%s	מחרוזת

מספר פרמטרים מוגדרים מראש בשפת **awk**.-כמה מהם מוצגים בטבלה 5.3.

טבלה 5.3 משתנים שמורים ב-awk

המשתנה	המטרה
FILENAME	שם קובץ הקלט הנוכחי
NR	מספר רשומת הקלט הנוכחי
FS	מפריד השדה (ראה אופצית F-)
NF	מספר השדות ברשומת הקלט הנוכחית
ORS	מחרוזת המפרידה בין רשומות הפלט

הפעלים המפורטים בטבלה 5.1 משמשים לחישובים אריתמטיים. האפשרויות להמרת תווים והמשתנים המפורטים בטבלאות 5.2 ו-5.3 מיועדים בעיקר להצגת הפלט המעובד עבור משתמש. ניתן להשתמש בפקודות **print** ו-**printf** להצגת פלט ערוך ולא ערוך בהתאמה (ראה דוגמאות **awk** בהמשך פרק זה).

מספר פונקציות של **awk** משמשות לטיפול בנתונים. שתיים מבין הפונקציות הנפוצות ביותר הן:

* **index(st2,st3)** מאתרת את המחרוזת "st3" במחרוזת "st2" ומחזירה את מיקומה. היא מחזירה את הערך אפס, אם לא ניתן למצוא את "st2". לדוגמה:

```
{ print index(name,"andy") }
```

* **substr(st,s,n)** מביאה n תווים המתחילים בנקודה s מתוך המחרוזת st; זוהי פונקצית תת-מחרוזת. לדוגמה:

```
{ (print substr(country,1,3) }
```

משתני awk והצבות

בנוסף למשתני **awk** שבטבלה **5.3**, יכול המשתמש להגדיר משתנים נוספים משלו. בכל משתנה ניתן להציב ערך מספרי או מחרוזת; כך, סוג המשתנה נקבע על ידי הערך שהוצב לו. בדוגמה למטה, המשתנה 'n' הינו משתנה מספרי והמשתנה 'y' הינו משתנה מחרוזת:

```
n=59
y="Nancy"
n=n + 3
```

תבניות תכנות מובנות

ב-**awk** כלולות מספר תבניות תכנות מובנות. משפט **if-else** ותבניות הלולאה **for** ו-**while** מתוארים בהמשך.

משפט if

```
if  (תנאי)
     משפט
  else
     משפט
```

לדוגמה, כדי לבדוק אם יש חמשה שדות בשורת קלט, ניתן לכתוב כך:

```
{ if($NF < 5)
      n = $NF - 1
  else
      n = 0
}
```

לולאת for

ללולאת **for** היא בעלת המבנה הבא:

```
for  (ביטוי1;תנאי;ביטוי2)
       משפט
```

כדי להציג את ארבעת שדות הקלט הראשונים בשורת קלט מסוימת, אפשר לכתוב כך:

```
for(n=1; n <= 4; n++)
   print $n
```

לולאת while

הלולאה **while** היא בעלת המבנה הבא:

```
while(תנאי)
   משפט
```

כדי להציג את ארבעת שדות הקלט הראשונים בשורת קלט מסוימת, אפשר לכתוב כך:

```
n=1
while(n <= 4){
     print $n
     ++n
}
```

דוגמאות לפקודות awk

המבנה הכללי של פקודת **awk** הוא:

```
$awk [-Fx] [ [ -f ] prog ] [filename]
```

האופציה **Fx-** מציינת שמפריד השדה הוא התו **'x'** ולא ברירות המחדל תו רווח, או תו טאבולטור. הארגומנט **prog** יכול להיות תכנית **awk** בת שורה אחת, או כאשר משתמשים באופציה **f-**, מציין הארגומנט **prog** קובץ המכיל תכנית מקור הכתובה בשפת **awk**. הארגומנט **filename** מציין את שם קובץ הנתונים שיש להשתמש בו כקלט לתכנית **awk**.

בהמשך נציג מספר דוגמאות של שימוש בהוראת **awk** שיבהירו את הרעיונות שהוסברו קודם.

דוגמה 1

הצג את תוכן הקבצים "ex4.c" ו-"temp.cbl".

```
$awk '{print}' ex4.c temp.cbl
```

דוגמה 2

מספר את כל שורות הקובץ "ex4.c":

```
$awk '{print NR, $0}' ex4.c
```

דוגמה 3

מספר את כל שורות הקובץ "ex4.c", כך שכל מספר שורה יהיה בן חמש ספרות:

```
$awk '{printf "%5d %s\n", NR, $0}' ex4.c
```

דוגמה 4

בחר את הטור הראשון והחמישי מהפלט של פקודת **who**. בהנחה שהפלט של הפקודה **who** הוא כדלקמן:

```
peter    tty02  Aug 16  14:15
andreas  tty08  Aug 16  08:12
rony     tty12  Aug 16  08:55
```

כאשר נכתוב את הפקודה

```
$who | awk '{printf "%s %s\n", $1, $5}'
```

נקבל את הפלט הבא:

```
peter    14:15
andreas  08:12
rony     08:55
```

דוגמה 5

המידע על התהליכים שפעילים כרגע במערכת מסופק על ידי הפקודה **ps**, נסה לבחון אם התהליך "master" הינו פעיל. אם הוא פעיל, הצב את מספר התהליך שלו למשתנה "pnum" ועצור אותו. שים לב, שהתווים שלפני הפקודה **ps** ולפני המשפט "if [$pnum=0]" הינם סימני טעם (grove accent) ולא מרכאות יחידות (כיוון התג הוא ` ולא ' בהתאמה).

```
pnum=`ps -e | awk '
    /master/ {
              found = 1;
              # print the process id of "master" which will be
              # assigned to the variable pnum
              print $1;
             }
    END      {
                      if (!found)
                             print "0";
                 }'`
if [ $pnum = 0 ]
then
```

```
          echo "The master program is not active\n"
else
          kill -9 $pnum
          case $? in
                    0) echo "The master program has now been
                              killed\n";;
                    *) echo "The master program has not been
                              killed\n";;
          esac
fi
```

דוגמה 6

השתמש בתכנית "staffsalary" כדי להפיק דו"ח על משכורות הצוות מתוך מידע הנמצא בקובץ "staffdata". השדה השני בכל שורה של הקובץ "staffdata" מייצג את המשכורת ומפריד השדה הינו תו נקודה-פסיק (;).

הצג את תוכן הקובץ "staffdata":

```
$cat staffdata
Thompson ;105000
Pilavakis;110000
Johnson  ; 95000
Werner   ; 90000
```

הצג את תוכן הקובץ "staffsalary":

```
$cat staffsalary
BEGIN { FS=";"}
{ total += $2}
END { print
     print
     print "SALARY REPORT"
     print "-----"
     print "Amount paid out = $" total
     print "Average salary = $" total/NR
     print "Number of employees processed = " NR}
```

פקודת **awk** שלהלן תפיק את הדו"ח המוצג אחריה:

```
$awk -f staffsalary staffdata

SALARY REPORT
-------------
Amount paid out                    = $400000
Average salary                     = $100000
Number of employees processed      = 4
```

5.2 חיפוש תבניות במספר קבצים

כאשר יש צורך לחפש תבנית (pattern) מסוימת בקובץ יחיד, ניתן להשתמש בעורך תמליל. אולם, כדי לחפש במספר קבצים, ניתן להשתמש בפקודה **grep**. פקודה זו תאתר תבנית תווים מסוימת מהר יותר וביעילות גבוהה יותר:

```
$grep [-אופציות] ביטוי [קבצים]
```

אופציות:

n- הצג בתחילת כל שורה את מספרה היחסי בקובץ.
v- הצג את כל השורות פרט לאלו שתואמות לביטוי.
c- הצג את מספר השורות (count) שבהן נמצאה התבנית.

לדוגמה, כדי לאתר את המחרוזת "static void" בקובץ "swap.c" השתמש בפקודה הבאה:

```
$grep -n "static void" swap.c
```

5.3 ספירת שורות

פקודת **wc** סופרת את מספר השורות, המלים או התווים בשמות הקבצים המצוינים, או בקלט הסטנדרטי, אם לא ניתנו שמות קבצים:

```
$wc [-אופציות] ביטוי [קבצים]
```

אופציות:

l - הצג את מספר השורות.
w - הצג את מספר המלים.
c - הצג את מספר התווים.

דוגמאות:

הצג את מספר המשתמשים שרשומים כרגע במערכת:

```
$who | wc -l
```

הצג את מספר השורות ואת מספר המלים בכל הקבצים בעלי ההרחבה (הציון) ".c".

```
$wc -lw *.c
```

5.4 השוואת שני קבצים

אפשר להשוות שני קבצים בעזרת הפקודה **cmp**. אם הקבצים זהים ולא צוינה אופציה כלשהי, לא ידווח דבר על המסך.

```
$cmp [-אופציות] קובץ1 קובץ2
```

אופציות:

l- דווח על המקום בקובץ (בבסיס עשרוני) שבו ישנו הבדל והצג את הבתים (בבסיס אוקטלי) השונים בכל קובץ.
s- החזר קוד 0 לקבצים זהים, 1 לקבצים שונים ו-2 לארגומנט חסר, או לארגומנט שלא ניתן להכנס אליו.

5.5 חלוקת קובץ לקטעים

ניתן לחלק קובץ למספר קטעים באמצעות הפקודה **split**:

```
$split [-אופציה/ות] קובץA קובץB
```

אופציות:

b- לציין קטעים בני 512 בתים
n- לציין את מספר השורות בכל קטע

הקבצים החדשים שנוצרים מקבלים את השם "קובץB" עם ההרחבה "aa", "ab" וכן הלאה עד ל-"zz". אם לא נרשם "קובץB", ברירת המחדל לשם הקובץ תתחיל עם "xaa", "xab" וכו'.

לדוגמה, כדי לחלק את הקובץ "master.dat" לקטעים בני 25 שורות כל אחד, השתמש בפקודות שלהלן. כך תבצע את החלוקה ותציג את שמות הקבצים:

```
$split -25 master.dat sec.dat
$ls
master.dat
sec.dataa
sec.datab
sec.datac
```

5.6 בחירת שדות מתוך קובץ

ניתן להשתמש בפקודה **cut** כדי לבחור שדות מתוך כל שורה בקובץ, או כדי לבחור טורים מתוך טבלה מסוימת. אם לא צויין שם קובץ, ישמש המסוף כקלט:

```
$cut -אופציות [שם קובץ]
```

אופציות:

-dx	לציין את "x" כתו מגביל (delimiter) שדה; הטאבולטור (tab) הינו ברירת המחדל של תו מגביל שדה.
-flist	לציין את שדות הבחירה.
-clist	לציין את מיקום התווים (לדוגמה, c1-15- בוחרת את 15 התווים הראשונים (1 עד 15).

בדוגמה שלהלן, השתמשנו בפקודה **cat** כדי להראות את תוכן קובץ הנתונים "ex.d". לאחר מכן נשתמש בפקודה **cut** כדי לבחור את הטורים (במלים אחרות, שדות 1 ו-3 מתוך "ex.d":

```
$cat ex.d
anne:22:female:programmer
andy:25:male:analyst

$cut -d: -f1,3 ex.d
anne:female
andy:male
```

5.7 תרגום של תווים

פקודת התרגום **tr** משמשת להעתקת תווים מהקלט הסטנדרטי אל הפלט הסטנדרטי תוך מחיקה או החלפה של התווים שמצויינים:

```
$tr [-אופציות] מחרוזתA מחרוזתB
```

אופציות:

- c- לתרגם את כל התווים, פרט לאלה שבמחרוזתA, לתווים המתאימים במחרוזתB.
- d- למחוק את כל תווי הקלט במחרוזתA.
- s- "לדחוס" את כל תווי הפלט שחוזרים על עצמם במחרוזתA, לתווים יחידים.

דוגמאות:

למחיקה של התוגים "ABCD" מתוך הקלט הסטנדרטי.נשתמש באופציה -d. המשתמש מקיש "ABCDefgh" ופקודת **tr** מציגה את "efgh" בלבד.

```
$tr -d ABCD
ABCDefgh
efgh
```

השתמש בפקודת **tr** עם האופציה -s כדי לדחוס את כל קבוצות המופעים. של תו הרווח לתו רווח יחיד. לאחר מכן השתמש בפקודת **cut** כדי להדפיס את השדה הראשון והשדה החמישי:

```
$who | tr -s ' *' | cut -d" " -f1,5
peter 14:15
andreas  08:12
rony 08:55
```

5.8 שילוב שורות מקבצים שונים

הפקודה **paste** יכולה לשמש לשילוב שורות מקבצים שונים, או לשילוב שורות עוקבות מקובץ מסוים:

```
$paste אופציות  שמות-קבצים
```

אופציות:

-   - לשמש במקום כל שם קובץ, כדי לקרוא שורה מהקלט הסטנדרטי.
- -dlist להחליף את מפריד השורה הסטנדרטי, שהינו תו הטאבולטור, בתווי "list".
- -s לשלב שורות עוקבות במקום שורה מכל קובץ קלט.

דוגמאות:

תחילה, השתמש בפקודת **cat** כדי לראות את תוכן הקבצים "file2" ו-"file3" ולאחר מכן שלב את שני הקבצים בעזרת פקודת **paste**:

```
$cat file2
this example
horizontal merging

$cat file3
shows the
of files

$paste file2 file3
this example shows the horizontal merging of files
```

כדי להציג רשימה של הקבצים במדריך הנוכחי בשלושה טורים ולהחליף את תו הטאבולטור ברווח, אפשר לכתוב כך:

```
$ls ¦ paste -d" " - - -
filea fileb filec
swap.c master.c data
```

5.9 היטל אוקטלי של קובץ

ניתן להציג היטל (dump) של קובץ בצורה אחת או יותר בעזרת פקודת **od** (octal dump). אם לא צויין שם קובץ, תשתמש הפקודה בקלט הסטנדרטי:

```
$od [-oxc] [שם קובץ]
```

אופציות:

o- להציג מלים בבסיס 8 (אוקטלי); זוהי ברירת המחדל.
x- להציג מלים בבסיס 16 (הקסדצימלי).
c- להציג מלים ב-ASCII.

בדוגמה שלהלן, משמשת הפקודה **od** להצגת הקובץ "infile", במבנה הערוך על פי בסיס 16 (הקסה-דצימלי) ועל פי ASCII:

```
$od -xc infile
0000000   5468  6973  2069  7320  616e  2065  7861  6d70
           T h   i s     i   s   a n     e   x a   m p
0000020   6c65  206f  6620  6120  6669  6c65  2062  6569
           l e     o   f     a     f i   l e     b   e i
0000040   6e67  2064  756d  7065  640a  7573  606e  6720
           n g     d   u m   p e   d \n  u s   i n   g
0000060   7468  6520  2d78  2020  616e  6420  202d  6320
           t h   e     - x         a n   d       -   c
0000100   206f  7074  696f  6e73  2e0a
             o   p t   i o   n s   . \n
```

5.10 מיון קבצים

ניתן למיין קובץ בעזרת פקודת **sort**:

$sort [-אופציות] [+1שדה] [-2שדה] [שמות קבצים]

אם צויין התו "-" במקום שם קובץ, ייקראו הנתונים מהקלט הסטנדרטי. התוצאות מפקודת **sort** ייכתבו לפלט הסטנדרטי.

אופציות:

m-	לשלב קבצים שמוינו בשלב מוקדם יותר.
tx-	להפריד השדה הוא התו **x**.
r-	להפוך את סדר המיון.
fieldA+	המספר הסדורי של המפתח הראשון.
fieldB-	המספר הסדורי של המפתח האחרון.

המיון יכול להיות מבוסס על מספר שדה מסוים, כאשר השדה הראשון (0+) הוא שדה מספר אפס. שדה הוא אוסף של תווים. ברירת המחדל של מפרידי השדות הם תווים ריקים ותווי

טאבולטור. אם לא צוינו בפקודה שדות, המיון יפעל על כל השורה.

בדוגמה שלהלן, הקובץ שעומד להיות ממוין, מודפס תחילה בעזרת פקודת **cat**. לאחר מכן הקובץ ממויין על פי השדה הרביעי בקובץ (שדה מספר 3 כאשר מתחילים לספור מ-0) שהוא מפתח המיון:

```
$cat infile
andy:113:100:Andy Jones:/usr/acct/andy:/bin/csh
rony:115:100:Rony Thornton:/usr/acct/rony:/bin/sh
guest:120:100:Guest:/usr/acct/guest:/bin/rsh

$sort -t: +3 -o outfile infile
$cat outfile
andy:113:100:Andy Jones:/usr/acct/andy:/bin/csh
guest:120:100:Guest:/usr/acct/guest:/bin/rsh
rony:115:100:Rony Thornton:/usr/acct/rony:/bin/sh
```



פרק 6

תכנית העריכה vi

במערכת UNIX טיפוסית יהיו בדרך כלל מספר תכניות עריכה כמו **vi**, **ed** ו-**ex**. ניתן להשתמש בתכניות עריכה אלו ליצירה ולשינוי של קבצים. אחת מתכניות העריכה הנפוצות ביותר היא **vi** (visual), שהיא תכנית עריכה חזותית, או מבוססת-מסך.

תכנית **vi** הופכת את מסך המסוף לחלון, שבו המשתמש רואה את הקובץ שברצונו לערוך. כל השינויים שבוצעו בקובץ מוצגים ישירות על המסך. השינויים גם מוכנסים לתוך מאגר (buffer), הנכתב על פי דרישה לדיסק כאשר מסתיימת העריכה.

כדי שאפשר יהיה לשנות קבצים, צריכה תכנית **vi** לדעת את סוג המסוף שמשתמשים בו. מזהה הסביבה משתנה-ערך, אשר כאן ערכו "שם מסוף=TERM", מודיע ל-**vi** את סוג המסוף שבשימוש. תיאור מלא של אפשרויות המסוף נמצא בבסיס הנתונים "terminfo", או בבסיס הנתונים "termcap". משתמש רגיל אינו צריך לדאוג לפרטים אלה, מכיון שמנהל המערכת אחראי לכך שבסיס הנתונים "terminfo", או בסיס הנתונים "termcap", יכילו את הנתונים הנכונים אודות המסוף.

6.1 תווים מיוחדים

תכנית העריכה מטפלת במספר תווים בצורה מיוחדת, לדוגמה:

* מקש CR (carriage return) - לסיום פקודות (לדוגמה: nu CR :set).
* המקשים DEL, או RUB - להפסקת פקודה (לדוגמה: \you DEL).
* המקשים ESC, או ALT - לסיום הכנסת טקסט (לדוגמה: O input-text ESC).
* Control-V - לביטול ההשפעה של תווים כמו רווחים, או CR (לדוגמה: Control-V CR).

6.2 יצירת קובץ

ניתן ליצור קבצים חדשים, או לבצע שינויים בקבצים קיימים,

בעזרת הפקודה הבאה:

```
$vi קבצים
```

לדוגמה, הפקודה

```
$vi ex5 temp.c
```

תאפשר למשתמש לערוך את הקבצים "ex5" ו-"temp.c". עם תחילת העבודה יוצב הסמן בתחילת הקובץ. השתמש בפקודה **i** כדי להכניס טקסט לפני הסמן וליצור את הקובץ. הכנסת טקסט מסתיימת בהקשת ESC במסוף. ניתן לשמור את השינויים בעזרת פקודת הכתיבה **:w** (או ZZ). לאחר שבוצעו כל השינויים הדרושים לקובץ "ex5" בדוגמה לעיל, תוכל הפקודה **:next** לאפשר לך לערוך את הקובץ "temp.c".

כדי לצאת מ-**vi** השתמש ב-**:q**. אולם, אם בוצעו שינויים בקובץ, פקודת **:q** לא תאפשר לצאת מ-**vi**. במקרה כזה הקש **:wq** כדי לשמור את השינויים ולצאת מ-**vi**, או שתקיש **:q!** כדי להתעלם מהשינויים שבוצעו בקובץ ולצאת מ-**vi**.

6.3 הזזת הסמן לאורך ולרוחב המסך

פקודות רבות של העורך מאפשרות להזיז את הסמן לכל נקודה על פני המסך:

מקש **return**	הזזת הסמן לתחילת השורה הבאה.
מקש **backspace**	הזזת הסמן תו אחד שמאלה.
מקש **space**	הזזת הסמן תו אחד ימינה.
j	הזזת הסמן שורה אחת למטה.
h	זהה ל-**backspace**.
l	זהה ל-**space**.
k	הזזת הסמן שורה אחת למעלה.
n+	הזזת הסמן n שורות קדימה (לדוגמה, +4).
n-	הזזת הסמן n שורות לאחור (לדוגמה, -4).
$	הזזת הסמן לסוף השורה.
0	הזזת הסמן לתחילת השורה (תו אפס).

ניתן למקם את הסמן בשורה מסוימת בקובץ על ידי כך שנצמיד לפני האות G מספר, שמציין את השורה הרצויה. הפקודה 12G למשל, תמקם את הסמן בשורה 12 בקובץ, ו-G בלבד - תמקם את הסמן בסוף הקובץ.

הפקודה CTRL-G תתן את מספר השורה שבה נמצא הסמן ואת שם הקובץ הנמצא כרגע בעריכה. הפקודות H, L ו-M מזיזות את הסמן בהתאמה לשורה הראשונה, האחרונה והאמצעית במסך. כדי לזוז

בקובץ מסך שלם, או חצי מסך, השתמש בפקודות הבאות:

- CTRL-F זוז קדימה מסך שלם.
- CTRL-D זוז קדימה חצי מסך.
- CTRL-B זוז אחורה מסך שלם.
- CTRL-U זוז אחורה חצי מסך.
- CTRL-Y חשוף שורה נוספת בחלק העליון של המסך.
- CTRL-E חשוף שורה נוספת בחלק התחתון של המסך.

6.4 חיפוש

מספר פקודות מאפשרות למשתמש לחפש מחרוזות בקובץ, או מלים ותווים בשורה. בזמן החיפוש ניתן להשתמש בתו '\' (backslash) כדי לבטל את המשמעות של תווים מיוחדים כמו '/', '?', '*' ו-']'. הפקודה **/john** תחפש קדימה, מהמקום שבו נמצא הסמן באותו רגע, את המחרוזת "john". אם נשתמש ב-'?' במקום '/' (לדוגמה, **?john**) אז החיפוש אחר המחרוזת "john" יתבצע לאחור. הפקודה **n** תבצע שוב את פקודת החיפוש ('/' או '?') האחרונה שבוצעה.

אפשר גם לחפש מלים ותווים בשורה. באופן כללי, מלה הינה מספר תווים אלפאנומריים המופרדים על-ידי רווח, או טאבולטור:

- **w** הזזת הסמן קדימה לתחילת המלה הבאה.
- **e** הזזת הסמן קדימה לסוף המלה הנוכחית.
- **b** הזזת הסמן אחורה לתחילת המלה הקודמת.

ניתן להצמיד לפקודות הרשומות לעיל מספר ואז, פקודה כמו **4b** תזיז את הסמן לאחור בארבע מלים. יש פקודות חיפוש מקבילות המשתמשות באותיות הגדולות **W**, **E** ו-**B**. הן מתייחסות לכל תו במלה, פרט לטאבולטור ורווח.

כדי לחפש תו מסוים בשורה השתמש בפקודות הבאות:

- **fn** חפש קדימה (מהמקום שבו נמצא הסמן) את התו **n**.
- **Fn** חפש לאחור את התו **n**.
- **;** חזור על פקודת **fn** או **Fn** האחרונה.

6.5 עריכת שורות תמליל

הפקודה המאפשרת להכניס תמליל (טכסט) לפני הסמן הוצגה בשלבים קודמים. הפקודה **O** (האות הגדולה O) תכניס תמליל לפני השורה

הנוכחית, עד אשר תתבצע הקשה על ESC, ואילו **o** (האות הקטנה **o**) תכניס תמליל לאחר השורה הנוכחית. יש פקודות נוספות אחרות המשמשות לעריכת שורות של תמליל:

a	הוסף תמליל לאחר הסמן (סיום בהקשת ESC).
A	הוסף תמליל בסוף השורה (סיום בהקשת ESC).
rx	החלף את התו הנוכחי בתו **x**.
dd	מחק את השורה הנוכחית.
dw	מחק מלה במקום שבו עומד כרגע הסמן.
x	מחק תו במקום שבו עומד כרגע הסמן.
X	מחק תו לפני המקום שבו עומד הסמן כרגע.

לפני הפקודות **dd**, **dw** ו-**x** ניתן להקיש מספר. לפיכך, הפקודה

3dd

תמחק שלוש שורות של טקסט, ואילו הפקודה

12x

תמחק 12 תווים מהמקום שבו עומד הסמן כרגע.

6.6 הזזה והעתקה של תמליל

למשתמש ב-**vi** יש מספר מאגרים (המסומנים בתווים **a** עד z), שניתן להשתמש בהם להעתקה והעברה של טקסט בתוך הקובץ או בין קבצים. תכנית העריכה עצמה משתמשת במאגר ללא שם לשם שמירת טקסט שעבר שינוי, או טקסט שנמחק.

הפקודה **Y** שומרת שורה לתוך המאגר ללא שם ו-**P** מחזירה אותו לקובץ לפני המקום שבו עומד הסמן כרגע. ניתן להעתיק מספר שורות לתוך מאגר בעזרת הפקודה

"nbY

סימן המרכאות הינו התו הראשון של הפקודה, **n** הינו מספר השורות שיש להעתיק, **b** הינו שם המאגר (יכול להיות כל תו מ-a עד z) ו-**Y** הינו קיצור של הפקודה **yy** המשמשת לשמירת השורות.

התבוננן ברצף הפקודות הבא:

12G	-	מקם את הסמן בשורה 12.
"8zY	-	העתק 8 שורות (מ-12 עד 20) לתוך מאגר z.
25G	-	מקם את הסמן בשורה 25.
"zp	-	הבא 8 שורות מתוך מאגר z ומקם אותן לאחר שורה 25.

באופן דומה, כדי למחוק חמש שורות מהמקום שבו נמצא הסמן כרגע

נכתוב את הפקודה

5dd

רצף טיפוסי אחר של פקודות שיכול לשמש למחיקת שורות הוא:

6dd - מחק 6 שורות ושמור אותן במאגר ללא שם.
23G - מקם את הסמן בשורה 23.
P - כתוב 6 שורות מתוך המאגר ללא שם ומקם אותן לפני שורה 23.

כדי להעתיק 87 שורות מהקובץ הנוכחי אל קובץ "temp", ניתן להשתמש בפקודות הבאות:

34G - מקם את הסמן בשורה 34.
"a87Y - העתק 87 שורות לתוך מאגר a.
:e temp - קרא את קובץ temp.
"ap - כתוב 87 שורות ממאגר a אל קובץ temp.
ZZ - שמור את קובץ temp.

6.7 פקודות חזרה וחרטה

ניתן לחזור על הפקודה האחרונה שהוקשה למסוף (repeat), בעזרת פקודת נקודה ('.'). לדוגמה, נשתמש בפקודה הבאה כדי לבטל 5 שורות

5dd

הפקודה שלהלן זהה לפקודה 5dd (שים לב ל- 4 הנקודות):

dd....

אפשר להתחרט על פעולה שנעשתה ולחזור בכל זמן מהשינוי האחרון (undo) בעזרת פקודת החזרה **u**. לדוגמה,

dd מחיקה של השורה הנוכחית מהקובץ.
u החזרת השורה המחוקה למקומה בקובץ.

פקודת האות הגדולה **U** תחזיר את השורה ותשחזר אותה למצבה המקורי, לפני שבוצעו בה שינויים כלשהם.

6.8 פקודות תכנית העריכה ed

שמת לבך בוודאי לכך שלפני מספר פקודות נכתבו נקודתיים (**:**). כל פקודה כזו היא למעשה פקודה של תכניות העריכה **ex** ו-**ed**, שניתן להשתמש בה בתכנית העריכה החזותית (**vi**). ניתן להשתמש

בתכנית העריכה **ed** במסוף-מסך או במסוף מסוג טלטייפ. מספר פקודות שימושיות של תכנית העריכה **ed** מתוארות בטבלה **6.1**.

שים לב שהסימן '^' מצביע על תחילתה של שורה. הנקודה (.) מציינת כל תו במהלך החיפוש. כמו כן יש לסיים כל פקודה ב-CR (carriage return).

טבלה 6.1 פקודות העורך ed שניתן להשתמש בהן בעורך vi

פקודות העורך ed	הפעולה
:y,zw שם-קובץ	כתוב שורות y עד z לקובץ (לפי "שם-קובץ").
:e שם-קובץ	קרא קובץ (לפי "שם-קובץ") לעריכה.
:e! שם-קובץ	התעלם מהשינויים הנוכחיים וקרא קובץ חדש.
:r שם-קובץ	כתוב קובץ לתוך המאגר.
:r !פקודה	כתוב את הפלט של הפקודה לתוך המאגר.
:n	ערוך את הקובץ הבא (קיצור ל-next).
:n!	התעלם משינויים בקובץ הנוכחי, ערוך את הקובץ הבא.
:/שם/	מצא בקובץ את המחרוזת "שם".
:/^שם/	מצא שורה המתחילה במחרוזת "שם".
:g/שם/s//שם2/gp	שנה את כל המופעים של המחרוזת "שם" למחרוזת "שם2".
:g/^..../s//שם/	שנה את ארבעת התווים הראשונים בכל השורות בקובץ למחרוזת "שם".
:s/שם2/שם3/	שנה את המחרוזת "שם2" למחרוזת "שם3".
:s/שם2/&and שם3/	שנה את המחרוזת "שם2" למחרוזת "שם2 ושם3".
:g/^[0-9]/s///gp	שנה את כל השורות המתחילות בספרה, מחק את הספרה והצג את כל השורות שבוצע בהן שינוי.
:.,$-5d	מחק את כל השורות מהשורה הנוכחית עד לחמש שורות לפני השורה האחרונה.
:/u.x/	מצא מחרוזת בת שלושה תווים המתחילה באות u ומסתיימת ב-x (לדוגמה, uax).
:1,$p	הדפס את כל הקובץ.
:12,20d	מחק שורות 12 עד 20.
:$	מקם את הסמן בסוף הקובץ.

6.9 קביעת אופציות

לתכנית העריכה יש קבוצה של אופציות, שניתן לקבוע אותן בעזרת פקודות בעלות המבנה הבא:

* **:set option** קביעת אופציה.
* **:set nooption** ביטול אופציה.

ניתן לקבל רשימה של כל האופציות שניתן להשתמש בהן ב-**vi** בעזרת הפקודה:

:set all

להלן מספר דוגמאות נוספות:

:set nu	מספר שורות בקובץ.
:set nonu	בטל את ההשפעה של פקודת **nu** הקודמת.
:set ai	הזזה (indentation) אוטומטית פנימה בתחילת פיסקה.
:set noai	בטל את פקודת **ai** הקודמת.

ניתן לקבוע אופציות אלו בצורה אוטומטית בכל פעם שנכנסים ל-**vi**, על-ידי השמה למשתנה EXINIT שבקובץ "$HOME/.profile":

```
EXINIT='set nu ai';export EXINIT
```

6.10 ביצוע פקודות מעטפת

ניתן לבצע פקודות מעטפת מתוך **vi** בעזרת הפקודה הבאה:

```
:!command return-key
```

לדוגמה, כדי להציג את כל המשתמשים הפועלים במערכת, נכתוב את הפקודה

```
:!who
```

אם יש לבצע מספר פקודות בזמן שמשתמשים ב-**vi**, ניתן ליצור מעטפת חדשה בעזרת הפקודה:

```
:sh return-key
```

לאחר פקודת "sh", ניתן לבצע מספר כלשהו של פקודות מעטפת. לאחר מכן יש להקיש על 'CTRL-D' כדי לחזור ל-**vi**. לדוגמה:

:sh return-key	מעבר למעטפת חדשה.
$who	הצגת המשתמשים המחוברים למערכת.
$ls	הצגת רשימת הקבצים במדריך הנוכחי.
CTRL-D	חזרה ל-vi.

פרק 7

עיבוד תמלילים

שלב התיעוד הינו אחד השלבים החשובים במהלך הפיתוח של כל מוצר. התיעוד הינו כלי תקשורת חשוב שמטרתו להעביר לאחרים, מנתחי מערכות ותכניתנים, את המידע אודות המוצר, כדי שיוכלו להשתמש במידע זה לצורך תחזוקה. למשתמשים שאינם תכניתנים, ישמש התיעוד ככלי עזר בהפעלת מוצר התוכנה בלי תמיכה ופיקוח של מומחי מחשב.
הערה: ההנחיות מכוונות לעבודה באנגלית, בכיוון שמאל-ימין.

ב-UNIX, ישנן מספר תכניות לעריכת תמלילים ("tbl", "eqn", "nroff", "troff") וחבילות מקרו ("mm", "ms") שניתן להשתמש בהן לתיעוד. פרק זה יתאר את עורך התמלילים "nroff" ואת חבילת המקרו "mm".

7.1 עורך התמלילים "nroff"

אחת התכונות החשובות ביותר של כל עורך תמלילים היא להפיק יישור שוליים מימין ומשמאל. ניתן לכתוב שורות פלט באופן חופשי, ואם המלה האחרונה בשורה אינה מגיעה לשוליים, בעת העריכה יתוספו בשורה רווחים, כדי ל"יישר" את השוליים. אפשר להשתמש במיקוף ו"לשבור" את המלה כדי שמספר הרווחים הנוספים יהיה קטן יותר. אפשרות המיקוף, כמו מאפיינים רבים של "nroff", הינם אופציונליים. הפקודה "nroff":

```
$nroff [אופציות] [קבצים]
```

תחילה, נשתמש בתכנית העריכה ליצירת הקובץ שיעובד על ידי "nroff". קובץ כזה יתבסס בדרך כלל על שורות תמליל ושורות של פקודות. שורות התמליל מכילות את הנתונים שיש לעבד ושורות הפקודה מציינות איך לערוך את הנתונים האלה. שורות הפקודה מתחילות בדרך-כלל בנקודה. התווים '\' ו-'"' משמשים לסימון שורות הערה ולכן העורך "nroff" מתעלם מהם. כפי שנראה בדוגמה להלן, ניתן לציין כקלט יותר מקובץ אחד. אם לא נרשם שם קובץ, תוכנת "nroff" תקרא את הקלט הסטנדרטי:

```
$nroff fileA.n fileB.n
```

7.2 פקודות נפוצות של "nroff"

במקרים רבים, פלט מ-"nroff" יודפס על דפי A4 סטנדרטיים. ניתן להשתמש למטרה זו בפקודת **ll.**, כדי לציין את אורך שורות הפלט:

```
.ll 70   \"  מציין אורך של 70 שורות
```

ניתן לעצור מילוי של שורה בעזרת פקודה לשבירת שורה:

```
.br
```

ניתן להפעיל או להפסיק מיקוף בעזרת הפקודות הבאות:

```
.hy 0   \"  הפסק מיקוף
.hy 1   \"  הפעל מיקוף
.hy a   \"  'a' קבע שתו המיקוף יהיה התו
```

שורות הפלט של "nroff" מודפסות בדרך כלל ברצף, ללא שורות רווח ריקות. הפקודה

```
.sp 3
```

תגרום ל-"nroff" להשאיר שלוש שורות ריקות בפלט. המרווח הסטנדרטי בין השורות יכול להקבע על ידי המשתמש:

```
.ls 2 \"  המרווח בין השורות הוא 2 שורות
```

אפשרות אחרת היא מרווח יחיד בין שורות הפלט:

```
.ss   \" .ls 1 מקביל לפקודה
```

מספר פקודות משנות את מספר התווים להזזת התמליל מגבול השוליים הרגיל (indent):

```
.in +5   \"  הזז פנימה 5 רווחים מהשוליים השמאליים
.in -5   \" . הזז החוצה 5 רווחים מהשוליים השמאליים
.in  5   \"  הזזה פנימה תתחיל בפוזיציה 5
```

הפקודה האחרונה (כלומר, **in 5.**) מאלצת את כל שורות הפלט להתחיל בפוזיציה 5. אולם ניתן להזיז פנימה שורה אחת בלבד, על ידי שימוש ב-"ti.":

```
.ti 15 \"  הזז פנימה ב-15 רווחים את השורה הבאה בלבד
```

בפקודה האחרונה משתמשים לעתים קרובות, כאשר פותחים פסקאות חדשות, שבהן מזיזים רק את השורה הראשונה של הפסקה.

אפשר להקדים את כל שורות הפלט של "nroff" במספר רווחים בעזרת הפקודה **po.** (page offset). הפקודה **po.** מבצעת לכל הקובץ (השורות) מה שהפקודה **ti.** מבצעת לשורה אחת:

```
.po 8   \"   הקדם את כל שורות הפלט ב-8 רווחים
.po     \"   החזר את המצב לקדמותו
```

ניתן לשנות את גודל הדף וניתן להתחיל דף חדש בכל מקום בפלט:

```
.pl 72\"   קבע שאורך הדף יהיה 72 שורות
.bp   \"   התחל דף חדש
.bp 3 \"   התחל דף חדש ומספר אותו כ-3
.he 7 \"   התחל דף חדש אם בדף הנוכחי אין מקום ל-7 שורות
.sk 2 \"   הוצא 2 דפים ריקים החל מהעמוד הבא
```

מילוי שורות הפלט נעשה על פי ברירת המחדל. אולם ניתן להפסיק זאת בעזרת הפקודה **nf.**:

```
.fi  \"   התחל מילוי
.nf  \"   הפסק מילוי של שורות הפלט
```

ניתן למרכז שורת פלט ו/או להוסיף לה קו תחתון. דבר זה נעשה בדרך כלל בכותרות:

```
.ce     \"   מרכז את שורת הקלט הבאה
.ce 2 , \"   מרכז את 2 שורות הקלט הבאות
.in +5 \"   הזז פנימה 5 רווחים מהשוליים השמאליים
.ul 2  \"   הוסף קו תחתון לשתי השורות הבאות
```

ניתן לציין אילו כותרות תופענה באופן קבוע בחלק העליון ובחלק התחתון של כל דף פלט מודפס, מבלי שיהיה צורך להכין אותן מחדש עבור כל דף. כותרות אלו נקראות "כותרות רצות". הפקודה הבאה קובעת כותרת עליונה (header title) רצה:

```
.he '1stpart'2ndpart'3rdpart'
```

התמליל "1stpart" מיושר לשמאל, התמליל "3rdpart" מיושר לימין ו-"2ndpart" הוא במרכז הדף. סימן % בכותרת מוחלף על ידי מספר הדף הנוכחי.

המבנה של פקודת הכותרת (**tl.**) דומה למבנה של הפקודה **he.**, כמו לדוגמה

```
.tl 'CAI-PACKAGE'-%-'CAI-PACKAGE'
```

פקודה זו תיישר לימין ולשמאל את המלה 'CAI-PACKAGE' ותמקם את מספר הדף באמצע שורת הפלט, כפי שנראה להלן (ללא המסגרת):

7.3 תכניות המקרו "nroff"

תכנית מקרו מתבססת על אוסף של פקודות "nroff". תכניות המקרו נמצאות בתחילת מסמכי "nroff", מכיוון שיש להגדיר אותן לפני שמשתמשים בהן. פקודת **de.** מתחילה כל מקרו ומגדירה את שמה. לאחר מכן ניתן להשתמש במספר פקודות "nroff" כדי ליצור את גוף המקרו. לבסוף, תכנית המקרו מסתיימת בשתי נקודות בשורה חדשה. הדוגמה שלהלן מראה את ההגדרה של תכנית המקרו "pg":

```
.de pg \"   השתמש בפקודת "de" כדי להגדיר מקרו בשם pg
.br    \"  שבירת שורה
.sp 2  \"  שתי שורות ריקות
.ti 10 \"   הזז את שורת הפלט הבאה בלבד ב-10 רווחים
..     \"  סיים את המקרו pg
```

ניתן להפעיל את המקרו לאחר מכן, בקובץ "nroff", על ידי ציון שם המקרו בתחילת שורה, כפי שנראה להלן:

```
.pg
```

אפשר גם להפעיל את המקרו בשורה מסוימת בפלט, על ידי שימוש בפקודת **wh.** (קיצור של when):

```
.wh  0 h2
.wh -7 f2
```

בדוגמה לעיל, הפקודה **wh.** הראשונה מציינת שיש להפעיל את המקרו של הכותרת הרצה העליונה (תכנית המקרו "h2" שהוגדרה על ידי המשתמש) בשורה אפס של דף הפלט. הפקודה **wh.** השניה מציינת שיש להפעיל את תכנית המקרו של הכותרת הרצה התחתונה ("f2") ולהדפיסה שבע שורות מתחתית כל דף.

ביחידות מחשב רבות המשתמשות ב-UNIX נוצר אוסף של תוכניות מקרו, אשר מתאימות לצורכי התיעוד המקומיים שלהם. מאוד מקובל להשתמש בפקודה **so.** כדי להכניס תכניות אלו לקובץ נפרד, שאותו ניתן לכלול במסמכים שעומדים לערוך. הנה דוגמה:

```
.so macrofile
```

בדוגמה זו, "macrofile" הוא שם הקובץ המכיל את תכניות המקרו. לקוראים המכירים את שפת התכנות C, נציין שהפקודה **so.**

דומה למשפט "#include <filename>" בתכנית C.

טבלה 7.1 מציגה את רוב הפקודות של "nroff".

טבלה 7.1 פקודות "nroff"

הפקודה	המשמעות
.ad	התחל ליישר את התמליל לשוליים הימניים. הצמד שורות לשוליים.
.ar	מספור שורות בשיטה העשרונית.
.br	שבור שורה. הפסק למלא את השורה הנוכחית.
.bl n	הכנס n שורות ריקות במקום הנוכחי.
.bp n	התחל דף חדש שמספרו n, ואם אין מציינים - מספרו 1.
.ce n	מספר את n השורות הבאות.
.de zz	הגדר מקרו הנקרא zz.
.ds	רווח כפול בין שורות הפלט. כמו 2 ls.
.ef tl	בדפים זוגיים הדפס את הכותרת התחתונה tl.
.eh tl	בדפים זוגיים הדפס את הכותרת העליונה tl.
.fi	התחל למלא את שורות הפלט.
.fo tl	כל הכותרות הרצות התחתונות הן tl.
.he tl	כל הכותרות הרצות העליונות הן tl.
.hx	בטל שורות כותרת.
.hy n	התחל מיקוף אם n=1 והפסק אותו אם n=0.
.in n	הזז n רווחים מהשוליים השמאליים. הזזת דף.
.ix n	זהה ל-'in.' אך בלי שבירה.
.ll n	אורך שורה הינו n תווים.
.ls n	המרווח בין השורות הינו n שורות.
.ml n	הכנס n שורות ריקות בין ראש הדף לבין הכותרת העליונה.
.na	הפסק ליישר תמליל לשוליים הימניים.
.ne	התחל דף חדש, אם אין מקום בדף ל-n שורות.
.nh	לא למקף.
.nn n	אין למספר את שורות הפלט הבאות.
.ni n	הזז מספרי שורה ימינה ב-n רווחים.
.nf	אל תמלא את שורות הפלט.
.nx filename	התחל לקבל קלט מ"שם קובץ". הפסק כאשר "שם קובץ" משתנה.
.nr y v	הכרז על אוגר y והצב בו את הערך v.
.of tl	הכותרת הרצה התחתונה בדפים אי-זוגיים הופכת להיות tl.
.oh tl	הכותרת הרצה העליונה בדפים אי-זוגיים הופכת להיות tl.

.pl n	אורך דף הינו n שורות.
.po n	התחל את כל השורות ב-n רווחים (page offset).
.ro	מספור דפים לפי השיטה הרומית.
.rm mname	הפסק את תכנית המקרו **mname** מכיון שאינה דרושה יותר.
.sk n	דלג n שורות ריקות בתחילת הדף הבא.
.so filename	הכנס את התוכן של "שם קובץ" במקום זה.
.sp n	דלג n שורות.
.ss	שורות הפלט במרווח יחיד.
.tc x	התו המחליף את הטאבולטור הופך להיות x (ברירת המחדל היא רווח).
.ti n	הזז את תחילת השורה הבאה בלבד ב-n רווחים.
.tl word	כותרת הדף היא word.
.ul n	הוסף קו תחתון ב-n השורות הבאות.
.wh n ex	הפעל את המקרו ex בשורה n.
..	פקודת סיום של הגדרת מקרו (מפסיקה את הפקודה .de).

7.4 עורך התמלילים "nroff" וחבילת התוכנה "mm"

בסעיף הקודם הוסבר איך להגדיר ואיך להשתמש בתכניות המקרו ב-"nroff". ב-UNIX ישנן מספר חבילות של תכניות מקרו שניתן להשתמש בהן בשילוב עם "nroff". הידועה ביותר היא חבילת "mm" (memorandum macro).

```
$nroff -mm [אופציות] [שמות קבצים]
```

הפקודה לעיל תגרום ל-"nroff" להפעיל גרסה לא דחוסה (non-compacted) של חבילה זו. אם מחליפים את **mm-** ב- **cm-**, "nroff" ישתמש בגרסה דחוסה של החבילה, אשר מאיצה את תהליך הטעינה.

פקודות "mm" שונות מפקודות רגילות של "nroff" בכך שיש לכתוב אותן באותיות גדולות לעומת פקודות "nroff" הנכתבות כולן באותיות קטנות. בדומה לפקודות "nroff" רגילות, גם פקודות "mm" מתבססות על שתי אותיות.

כדי לעזור בתהליך הלימוד, השתמשנו בסדרה של דוגמאות הנמצאות בתת-הסעיפים הבאים. רוב הדוגמאות מתבססות על שני חלקים; קובץ קלט, אשר מכיל את התמליל ואת הפקודות שחבילת "nroff" צריכה לעבד, וקובץ פלט אשר מציג את התוצאה של העיבוד של קובץ הקלט.

אוגרים מספריים

ניתן להשתמש בפקודת **nr.** (number register) כדי להכריז על אוגרים חדשים, או לשנות כאלה שהוגדרו קודם. שמות אוגרים יכולים להיות באורך של עד שני תווים:

```
.nr k 2
```

הפקודה תכריז על אוגר k ותקצה לו את הערך 2. ניתן להדפיס את הערך של האוגר k על ידי כך שנוסיף לפניו את התווים "n\". לדוגמה,

```
.sp \nk
```

הפקודה הזו תדפיס (תדלג) שתי שורות ריקות, מכיון שהאוגר k אותחל בערך 2.

ניתן להשתמש באוגר HB (head break), כדי לציין את רמת הכותרות שבעקבותיהן תבוא שבירה. לדוגמה,

```
.nr HB 7
```

הפקודה לעיל מציינת שבעקבות כותרות מרמה 1 עד 7 צריכה לבוא שבירה. ברירת המחדל היא שרק בעקבות כותרות מרמות 1 ו-2 צריכה לבוא שבירה. הפקודה

```
.hr HS 7
```

מציבה באוגר "רווח כותרת" (head space) את הערך 7, כדי שתהיה שורה ריקה לאחר כל הכותרות העליונות. הפקודה

```
.nr EJ 2
```

מציבה ערך באוגר "קפיצת" הדף (ejection) וכך, כל הכותרות מרמה 1 ו-2 יתחילו בדף חדש. הפקודה

```
.nr CL 7
```

מציבה את הערך 7 באוגר רמת התוכן (content level), כך יופק תוכן עניינים המכיל את כל שבע הרמות של הכותרות.

(התייחס בהמשך לסעיף בפרק זה הנושא את השם "כותרות ממסופרות ותוכן העניינים", שם השתמשנו בפקודות שהוסברו לעיל).

פקודות נוספות לעריכת הדף

הפקודה **SP** של "mm" (אשר זהה לפקודה **SP** של "nroff") מפיקה (או מדלגת) שורה ריקה. ניתן להשתמש במספר כדי לפרט כמה

שורות ריקות רוצים לדלג. לדוגמה,

.SP 2

פקודה זו מדלגת שתי שורות. הפקודה הבאה

.SK 3

מדלגת על שלושה דפים. כדי לדלג על דף אחד נכתוב את הפקודה

.SK

ניתן להפעיל יישור שוליים מימין על ידי שימוש בפקודה

.SA 1

ולהפסיק יישור שוליים על ידי הפקודה

.SA 0

פקודת List

הפקודה List משמשת להדגשת פריטים ברשימה. היא בעלת המבנה הבא:

.LI [סימן] [כל תו]

האופציה **"סימן"** (mark) מציינת את התו שישמש להדגשת הפריט. האופציה **"כל תו"** (any-char) יכולה להיות כל תו והיא משמשת לציון העובדה שהסימן חייב לבוא לפני כל ספרה מודפסת (ספרות עשרוניות בדוגמה הבאה).

קובץ הקלט

```
.SA 1
This example demonstrates the use of the List command for
emphasizing particular items.
.AL
.LI ! z
This item is marked with the "!" symbol
.LI
This item is not marked with "!"
.LI ! z
Again this item is emphasized by the "!" mark
.LE
```

```
This example demonstrates the use of the List command for
emphasizing particular items.

! 1.  This item is marked with the "!" symbol

  2.  This item is not marked with the "!"

! 3.  Again this item is emphasized by the "!" mark
```

הפקודת List End

כל פקודות הרשימה (list commands) חייבות להסתיים בפקודה LE., אשר המבנה שלה הוא:

.LE [שורות ריקות]

האופציה "שורות ריקות" יכולה לקבל את הערכים 0 (ברירת מחדל) ו-1, כאשר 0 פירושו בלי שורות ריקות ו-1 פירושו הוספת שורה ריקה אחת לאחר הרשימה.

הפקת רשימה באופן אוטומטי

הפקודה AL. (automatic list) מפיקה באופן אוטומטי רשימות שממוספרות בצורה שצוינה בפקודה. מבנה הפקודה:

.AL [סוג] [הזזה] [שורות ריקות]

האופציה **סוג** יכולה לקבל אחד מהערכים הבאים:

1	ספרות עשרוניות (ברירת מחדל).
i	אותיות רומיות קטנות.
I	אותיות רומיות גדולות.
a	אותיות אנגליות קטנות.
A	אותיות אנגליות גדולות.

האופציה **הזזה** מציינת את ערך ההזזה (indentation). אם האופציה **שורות ריקות** מקבלת את הערך 1, אז הפריטים יוצגו ברשימה ללא שורה ריקה שמפרידה ביניהם. אם האופציה **שורות ריקות** היא 0 (ברירת המחדל), אז בין כל פריט ופריט ברשימה תפריד שורה ריקה.

אם לא צויין ערך **להזזה**, יהיה ערך זה שווה לערך הנמצא באוגר **Li**. (אוגר רמת ההזזה). ניתן להשתמש באוגר **Ls**. (אוגר רמת הרווח) כדי לציין באיזה רמות יש להשאיר שורה ריקה מעל הפריט ברשימה ומתחתיו. לדוגמה,

.Ls 2

פקודה זו מציינת שיש להשאיר שורה ריקה לפני ואחרי כל פריט הנמצא ברמות 1 ו-2. הדוגמה שלהלן מראה את השימוש בפקודה **AL**.

קובץ קלט

```
.SA 1
.nr LS 2
.AL
.LI
This is level one.  The example shows that .AL commands can be
nested.
.AL a
.LI
This second level item is labelled "a".  Subsequent items will be
labelled with letters until .LE is encountered.
.LI
This item is labelled "b".
.LE
.LI
Now that the second level has been ended with the .LE command.
Level one starts again until next .LE command.
.LE
```

קובץ פלט

```
1.  This is level one.  The example shows that  .AL commands
    can be nested.

    a.  This second level item is labelled "a".
        Subsequent items will be labelled with letters
        until .LE is encountered.

    b.  This item is labelled "b".

2.  Now that the second level has been ended with the  .LE
    command.  Level   one   starts  again  until next  .LE
    command.
```

פקודת Dash List

פקודה **Dash List** היא בעלת המבנה הבא:

.DL [הזזה] [שורות ריקות]

האופציה **הזזה** מציינת את ערך ההזזה פנימה (10 רווחים בדוגמה שלהלן). אם האופציה **שורות ריקות** שווה ל-0, אז שורה ריקה אחת תפריד בין הפריטים ברשימה, ואם הערך שווה ל-1 אז אף לא שורה אחת תפריד בין הפריטים. הדוגמה שלהלן מציגה את השימוש בפקודת **DL.**.

קובץ הקלט

```
.SA 1
.nr LS 2
This example demonstrates the Dash List where each item is
indented ten spaces and marked with a dash.
.DL 10
.LI
This is the first item
.LI
This is the second item
```

קובץ הפלט

```
This example demonstrates the Dash List where each item is
indented ten spaces and marked with a dash.

          - This is the first item

          - This is the second item
```

פקודת קו תחתון

ניתן להשתמש בפקודה **I.** יחד עם הפקודה **R.** כדי לשלוט בשרטוט קו תחתון מתחת לתמליל.

כאשר משתמשים בפקודה .I עם ארגומנט יחיד, ימתח קו תחתון רק מתחת לאותו ארגומנט. ללא ארגומנט, יהיה קו תחתון מתחת לכל התמליל שרשום אחריה, עד שתתקל בפקודה .R . להלן דוגמה:

קובץ הקלט

```
.SA 1
The .I will underline a single argument that follows
.I it
If no argument is provided, then .I will
.I
underline all text on following Lines until a .R
.R
command is encountered.
```

קובץ הפלט

The .I will underline a single argument that follows it. If no argument is provided, then .I will underline all text on following lines until a .R command is encountered.

כותרות ממוספרות ותוכן עניינים

ניתן למספר כותרות באופן אוטומטי ולערוך אותן באמצעות תכניות המקרו הנמצאות בחבילת "mm". בגמר התמליל ניתן להפיק באופן אוטומטי תוכן עניינים, שיכול להכיל עד שבע רמות של כותרות. עושים זאת בעזרת הפקודה .TC.
המבנה של פקודת הכותרת .H הוא:

.H רמה [תוכן הכותרת]

הדוגמה שלהלן מציגה את השימוש בכותרות ובפקודה .TC:

קובץ הקלט

```
.nr CL 7
.nr HB 7
.nr HS 7
.SA 1
.H 1 "Computer Aided Instruction Package"
```

פקודת Dash List

פקודה Dash List היא בעלת המבנה הבא:

.DL [הזזה] [שורות ריקות]

האופציה **הזזה** מציינת את ערך ההזזה פנימה (10 רווחים בדוגמה שלהלן). אם האופציה **שורות ריקות** שווה ל-0, אז שורה ריקה אחת תפריד בין הפריטים ברשימה, ואם הערך שווה ל-1 אז אף לא שורה אחת תפריד בין הפריטים. הדוגמה שלהלן מציגה את השימוש בפקודת DL..

קובץ הקלט

```
.SA 1
.nr LS 2
This example demonstrates the Dash List where each item is
indented ten spaces and marked with a dash.
.DL 10
.LI
This is the first item
.LI
This is the second item
```

קובץ הפלט

```
This example demonstrates the Dash List where each item is
indented ten spaces and marked with a dash.

          - This is the first item

          - This is the second item
```

פקודת קו תחתון

ניתן להשתמש בפקודה I. יחד עם הפקודה R. כדי לשלוט בשרטוט קו תחתון מתחת לתמליל.

```
This example demonstrates the use of numbered headings. Headings
are commonly used for documentation because a table of contents
may be produced with the .TC command.
.H 2 "CAI questions"
The CAI package can handle three types of questions.
.H 3 "Multiple Choice questions"
The user has to select correct assertions.
.H 3 "Yes-or-no questions"
The user has to reply with a yes or a no.
.H 3 "True-or-false questions"
The user has to give a true or false answer.
.H 2 "CAI statistics"
This is a superuser facility that assesses the performance
of the users.
.TC
```

קובץ הפלט

1. <u>Computer Aided Instruction Package</u>

This example demonstrates the use of numbered headings. Headings are commonly used for documentation because a table of contents may be produced with the .TC command.

1.1 <u>CAI questions</u>

The CAI package can handle three types of questions.

1.1.1 <u>Multiple Choice questions</u>

The user has to select correct assertions.

1.1.2 <u>Yes-or-no questions</u>

The user has to reply with a yes or a no.

1.1.3 <u>True-or-false questions</u>

The user has to give a true or false answer.

1.2 <u>CAI statistics</u>

This is a superuser facility that assesses the performance of the users.

CONTENTS

כותרות רצות עליונות ותחתונות

כותרת עליונה רצה וכותרת תחתונה רצה יכולות להופיע בהתאמה בראש כל דף ובתחתיתו. המבנה של הכותרת העליונה הוא כדלקמן:

```
.PH "'part1'part2'part3'"
```

התמליל "part1" מוצמד לשמאל, התמליל "part3" מוצמד לימין, ו-"part2" במרכז.

לפקודות שלהלן יש מבנה זהה למבנה הפקודה PH.:
PF. לכותרת תתחתונה רצה, EH. לכותרת עליונה בדפים זוגיים, ו-OH. לכותרת עליונה בדפים אי-זוגיים. התבנית

\\\\nP

בתוך כותרת יכולה לשמש להפקת מספר הדף, כפי שנראה להלן:

קובץ הקלט

```
PH "'Left-part'center-part'Page \\\\nP'"
This demonstrates the printing of a running header and page
numbering.
```

קובץ הפלט

```
Left-part                 center-part                 Page 1
This demonstrates the printing of a running header and page
numbering.
```

פרק 8

ביצוע תכנית בשפת C

בפרק זה נלמד כיצד לבצע תכנית בשפת C, אשר הוכנה בעזרת תכנית עריכה (ראה פרק 6). נתרכז בעיקר בפקודה **cc** המשמשת להידור ולקישור (link) של תכניות C ומייצרת קובץ פקודות שניתן להרצה (executable file).

הנושאים שנדון בהם כוללים הידור קבצי מקור, קדם המעבד של שפת C, מבנה קבצי הרצה, איך להקטין את גודל הקבצים הללו כדי להגביר את מהירות הביצוע וכיצד להכין במעטפת קבצי פקודות להידור של תכניות.

8.1 הידור של קובץ מקור

הידור של תכנית בשפת C, אשר נערכה ונשמרה עם הסיומת ".c", נעשה בפקודה **cc**:

```
$cc [אופציות] [שמות קבצים]
```

(כרגיל, הסוגריים המרובעים מציינים שהארגומנטים הינם אופציונליים).

לאחר שתכנית עברה הידור, נשאר קובץ היעד (object) שלה על הדיסק באותו שם של קובץ המקור, אבל הסיומת ".c" משתנה ל-".o". באופן רגיל קובץ ".o" מתבטל, מכיון שפקודת **cc** מפעילה את תכנית הקישור המפיקה קובץ ביצוע ונותנת לו, כברירת מחדל, את השם "a.out" (בהנחה שאין טעויות תחביריות). כדי לבצע את "a.out" הקש

```
$a.out
```

ניתן לשנות את ברירת המחדל "a.out" לכל שם אחר בעזרת האופציה **-o**. לדוגמה, אם תכנית מקור נקראת "test.c" ואנו רוצים ששם תכנית הביצוע (executable program) יהיה "test.e" במקום "a.out", נשתמש בפקודה זו:

```
$cc -o test.e test.c
```

ניתן ליצור קובצי יעד בעזרת האופציה **-c**. אופציה זו מנחה את המהדר להפיק קובצי יעד עם הסיומת ".o". לא יופקו קובצי

ביצוע, מכיון שלא ניתנה הוראה לבצע את שלב הקישור שלאחר ההידור. לדוגמה, כדי להפיק את קובץ היעד "math.o", נשתמש בפקודה:

```
$cc -c math.c
```

8.2 הידור של מספר קבצים

תכניות גדולות מחולקות לעתים למספר קבצים כדי להקל על הניהול שלהן. במקרים כגון אלה, רק אחד מבין הקבצים צריך להכיל את הפונקציה 'main'. בפקודה זו, לדוגמה, הקבצים "math.c", "add.c" ו-"mult.c" הינם קובצי מקור הכתובים בשפת C. תכנית הביצוע היא final.e:

```
$cc -o final.e math.c add.c mult.c
```

ניתן לבצע את הפקודה cc עם מספר קובצי מקור וקובצי יעד, או עם קובצי יעד בלבד. לדוגמה פקודה, אשר מפיקה תכנית לביצוע "final.e" מקובץ המקור "math.c" ומקובצי. היעד "add.o" ו-"mult.o":

```
$cc -o final.e math.c add.o mult.o
```

ניתן להפיק קובץ מקור הכתוב בשפת אסמבלי בעזרת האופציה S-. נשתמש לדוגמה בפקודה הבאה:

```
$cc -S add.c
```

היא תפיק קובץ מקור "add.s", הכתוב בשפת אסמבלי. הפקודה

```
$cc -o final.e math.c add.s math.o add.o
```

תבצע הידור של "math.c", תבצע אסמבלי של "add.s" ותקשר את "math.o" ו-"add.o" כדי ליצור את התכנית לביצוע "final.e".

8.3 קדם המעבד ב-C והידור מותנה

פקודת cc מספקת מספר אופציות, אשר מאפשרות לשלוט על הפעולות המתבצעות על-ידי קדם המעבד של שפת C.

קבצים מוכללים

כל תכנית C המשתמשת באמצעי קלט/פלט סטנדרטים חייבת לכלול בתחילתה את ההוראה

```
#include <stdio.h>
```

הוראה זו מורה לקדם המעבד של C להכליל בנקודה זו את המשפטים הנמצאים בקובץ "/usr/include/stdio.h". הסימנים < ו-> מציינים את המדריך הסטנדרטי שהינו בדרך כלל "/usr/include".
כאפשרות נוספת אפשר להשתמש בהוראה

```
#include "/usr/john/table.h"
```

.

היא תגרום לכך שבתכנית C יוכללו המשפטים הנמצאים בקובץ "table.h" שבמדריך "/usr/john".

בעזרת האופציה I- בזמן ההידור ניתן להגדיר בצורה מפורשת את המדריכים המכילים קבצים מוכללים. נשתמש בפקודה

```
$cc -I/usr/john test.c
```

היא תגרום לכך שההוראה שלהלן, הנמצאת בקובץ "test.c",

```
#include <table.h>
```

תוחלף בתוכן הקובץ "/usr/john/table.h".

החיפוש בברירת המחדל של המדריך "/usr/include" יתבצע רק לאחר חיפוש במדריכים שצוינו עם האופציה I.

הידור מותנה

הידור מותנה ניתן לבצע בעזרת האופציה D-. זהו מאפיין נוסף שמספק קדם המעבד C, אשר מאפשר למשתמש לציין אילו שורות בתכנית יש להדר ומאילו יש להתעלם.

אפשר להשתמש באופציה D- כדי להגדיר ערך של מקרו. נוח לעשות זאת כאשר משתמשים בהוראות **#if**, **#ifdef**, **#ifndef**, **#else** ו-**#endif**. אופציה זאת היא בעלת המבנה הבא:

```
-Dmname [=mvalue]
```

"mname" הינו שם המקרו ו-"mvalue" הינו הערך של המקרו. אם לא צויין ערך למקרו, מניחים שהמקרו מוגדר עם הערך 1 (אמת).

עיין בקטע הבא מתוך התכנית "test.c"

```
#ifdef CTRACE
     printf("Entered routine mult\n");
     printf("Value of count=%d\n",count);
#endif
```

נשתמש בפקודה

```
$cc -DCTRACE test.c
```

כדי להדר את כל ההוראות בין **#ifdef** לבין **#endif**. על ידי כך ניתן יהיה לבצע את שתי ההוראות **printf** בזמן הביצוע של התכנית. הפקודה שלפנינו,

```
$cc test.c
```

תתעלם מההוראות הנמצאות בין **#ifdef** לבין-**#endif**, מכיון שהאופציה **D-** לא צוינה.

אם רוצים להדר מספר קבצים, יש לציין את האופציה **D-** לפני שם הקובץ שבו יש להגדיר את המקרו. לדוגמה, נשתמש בפקודה

```
$cc mult.c -DCTRACE add.c
```

כדי להגדיר את המקרו "CTRACE" עבור "add.c" בלבד. בדרך זו, לאופציה "DCTRACE-" תהיה השפעה על "add.c", כמו להוראה

```
#define CTRACE 1
```

שבתוך הקובץ "add.c".

8.4 מבנה התכנית

בהנחה שאין שגיאות תחביריות, תפיק תכנית הקישור קובץ לביצוע בשם "a.out". בטבלה 8.1 ניתן לראות כיצד קובץ כזה מאורגן. כאשר נפעיל את פקודת **cc** עם האופציה **g-**, יפיק המהדר בקובץ היעד תוספת מיוחדת עבור כל שורת מקור של C, שבה קיימת אפשרות לציין נקודת שבירה (breakpoint). משתמשים יכולים לקבוע נקודות שבירה במספרי השורה המתאימים בעזרת תכניות ניפוי כמו "sdb", או "abb". ניתן למצוא את מבנה התוספת המיוחדת של מספרי שורה בקובץ

```
/usr/include/linenum.h
```

נתוני הניוד (relocation) אינם דרושים, אם לאחר הקישור לא יהיו שמות חיצוניים שיש לפתור (unresolved). המידע לגבי טבלת המחרוזות לא יהיה קיים אם קובץ המקור אינו מכיל שמות ארוכים יותר משמונה תווים. כמו כן, אם תכנית "הושארה במערומיה" בעזרת האופציה **s-** (ראה בסעיף הבא), לא יהיה מידע לגבי נתוני הניוד, מספרי השורה, טבלת השמות וטבלת המחרוזות.

מידע על כותרת הקובץ	**FILE HEADER INFO**
כותרת הקובץ	File Header
כותרות עזר	Auxiliary Headers
כותרת text	.text section header
כותרת data	.data section header
כותרות bss	.bss section headers
מידע על הנתונים	**DATA INFO**
נתוני text	.text section data
נתוני data	.data section data
מידע על נתוני הניוד	**RELOCATION DATA INFO**
נתוני הניוד של ה-text	.text section relocation data
נתוני הניוד של ה-data	.data section relocation data
מידע על מספרי שורה	**LINE NUMBER INFO**
מספרי השורה של ה-text	.text section line numbers
מספרי השורה של ה-data	.data section line numbers
מידע על טבלת הסימנים	**SYMBOL TABLE INFO**
סמלי ה-text	.text section symbols
סמלי ה-data	.data section symbols
סמלי ה-bss	.bss section symbols
מידע על טבלת המחרוזות	**STRING TABLE INFO**
שמות מחרוזות	string names

תרשים 8.1 מבנה של קובץ יעד

חלק ה-'text.' (תמליל/טכסט) מכיל הוראות לביצוע, חלק ה-'data.' מכיל משתנים שניתן להם ערך התחלתי ואילו 'bss.' מכיל משתנים ללא ערך התחלתי. לדוגמה, נתונות ההכרזות הבאות אשר נמצאות בתכנית C:

```
int i=10;
int y;
```

המשתנה 'i' ימוקם בחלק הנקרא 'data.' ו-'y' - בחלק הנקרא 'bss.'.

תכניות לביצוע המופקות על ידי תכנית קישור יכולות להיות שייכות לאחת משתי קטגוריות: תכניות עם 'סגמנט טקסט לא טהור' (impure text sagment) ותכניות עם 'סגמנט טקסט טהור' (pure

(text segment). במקרה של תכניות עם סגמנט טקסט לא טהור, יוצרים הסגמנטים של הטקסט והנתונים סגמנט אחד ויחיד לקריאה וכתיבה. לתכניות עם סגמנט טקסט טהור יש סגמנט טקסט משותף לקריאה בלבד וסגמנט נתונים לקריאה וכתיבה (ראה תרשים 8.2).

סגמנט מחסנית לקריאה/כתיבה Read/Write Steak Segment
סגמנט נתונים לקריאה/כתיבה Read/Write Data Segment
סגמנט טקסט לקריאה בלבד Read-only Text Segment

תרשים 8.2 מרחב כתובות של תכנית עם סגמנט טקסט טהור

כאשר קובץ ביצוע נטען לזיכרון, נוצרים שלושה סגמנטים לוגיים: סגמנט טקסט, סגמנט נתונים (מתבסס על נתונים שאותחלו ולאחריהם נתונים שלא אותחלו) וסגמנט מחסנית.

הכותרת (ראה תרשים 8.1), אשר לעולם אינה נטענת לזיכרון, נבדקת על-ידי הגרעין. אם השדה הראשון, אשר ידוע בשם "מספר הקסם" (magic number) מכיל את הערך 410 (בבסיס 8), אזי התכנית נחשבת 'סגמנט טקסט טהור'. המשמעות היא שאם תהליכים שונים מבצעים את אותה התכנית, הם משתמשים בסגמנט טקסט יחיד. כלומר, הם משתמשים באותו עותק של התוכנית, אולם לכל אחד מהם יהיה עותק נפרד של הנתונים. אם מספר הקסם בכותרת יכיל את הערך 407 (בבסיס 8), תחשב התכנית כתכנית עם 'סגמנט טקסט לא טהור'.

סגמנט הנתונים הוא בדרך כלל סטטי, אך הוא יורחב אם יתבצעו קריאות ל-"malloc", או ישירות לשגרות "brk" או "sbrk".

סגמנט המחסנית, אשר משמש לשמירת מידע, כמו משתנים מקומיים של הפונקציה וכתובות חוזרות, יכול להתרחב בצורה דינמית.

אופציות המהדר הדרושות להפקת סגמנט טקסט טהור או סגמנט טקסט לא טהור, משתנות ממהדר למהדר. מספר מהדרים מפיקים כברירת מחדל תכניות עם סגמנט טקסט טהור ויש צורך להשתמש באופציה N- כדי להפיק תכניות עם סגמנט טקסט לא טהור. מהדרים אחרים משתמשים באופציה **i**- ובאופציה **n**- כדי להפיק תכניות עם סגמנט טקסט טהור.

במספר מערכות UNIX, יכולים המהדרים להפיק 'מודל בינוני' של קובץ לביצוע או 'מודל קטן' שלו. במערכות אלו, אינך יכול

להפיק תכנית עם סגמנט טקסט טהור אם המהדר מפיק מודל בינוני של תכנית. אולם, אין להתעלם מהיתרונות של תכניות עם סגמנט טקסט טהור המתבטאים במהירות עיבוד ובחיסכון של מקום בזיכרון.

8.5 אופטימיזציה של מהירות הביצוע

בזמן ההידור ניתן להפעיל את תכנית האופטימיזציה בעזרת האופציה O-. בדרך זו אפשר לקבל תכנית קטנה יותר שתתבצע מהר יותר. יש להשתמש באופציה זו בשלב האחרון של פיתוח התכנית, לאחר שנופתה משגיאות. כדי לבצע אופטימיזציה לקבצים "math.c" ו-"mult.c" וליצור קובץ ביצוע אופטימלי הנקרא "a.out" נשתמש בפקודה זו:

```
$cc -O math.c mult.c
```

כדי להפיק קובץ יעד אופטימלי הנקרא "math.o" נשתמש בפקודה:

```
$cc -O -c math.c
```

ניתן להקטין את תכנית הביצוע במידה רבה עוד יותר על ידי שימוש באופציה s- (strip). אופציה זו מבטלת את מספרי השורה ואת המידע בטבלת השמות (ראה תרשים 8.1), אשר דרושים לתכניות לניפוי שגיאות, כמו "adb" ו-"sdb". לשם כך נשתמש בפקודה

```
$cc -s math.c mult.c
```

פקודה זו תפיק תכנית ביצוע שאינה מכילה מספרי שורות ולא מידע על טבלת השמות. ניתן להשיג את אותה תוצאה על ידי שימוש בפקודה **strip** של UNIX.

האופציות O- ו-s- יעילות במיוחד בזמן ההתקנה במערכות שבהן מרכיב הזמן ומרכיב המקום מהווים בעיה.

8.6 יצירת ספריות וקישור עצמים מספריות

הפקודה **ar** יכולה לשמש לתחזוקת קבוצה של קבצים או פונקציות בספריה יחידה, אשר מתייחסים אליה כקובץ ארכיון. קובץ ארכיון יכול להכיל, לדוגמה, את כל השגרות הקשורות לפרויקט מסוים ואשר דרושות למודולי מקור שונים של אותו פרויקט. כל קבצי היעד לשגרות סטנדרטיות בשפת C וקריאות מערכת נמצאות בקובץ הארכיון "/lib/libc.a".

בדוגמה שלהלן, יוצרים את הספריה "libkalpha.a" שבתוכה קובצי היעד "add.o", "mult.o" ו-"sub.o".

```
$ar cr libkalpha.a add.o mult.o sub.o
```

כדאי (מסיבות שיוסברו מאוחר יותר), שכל שם של ספריה יתחיל בשם 'lib' ויסתיים ב-'.a'. האופציה **v** (verbose) מבטיחה ש-**ar** תדפיס את הפעולות שהיא מבצעת. כדי להדפיס את תוכן הספריה שזה עתה נוצרה, נשתמש בפקודה

```
$ar t libkalpha.a
```

כדי לשלוף את קובץ היעד "mult.o" מהספריה נשתמש בפקודה

```
$ar xv libkalpha.a mult.o
```

כדי להחליף בספריה את קובץ "mult.o" בגרסה מעודכנת יותר נשתמש בפקודה

```
$ar rv libkalpha.a mult.o
```

כדי להעביר את קובץ "mult.o" כך שיהיה לפני "add.o", נשתמש בפקודה

```
$ar mvb add.o libkalpha.a mult.o
```

נשתמש באות **b** כדי לציין "לפני" וב-**a** כדי לציין "אחרי".

לאחר שנוצרו ספריה או קובץ ארכיב, נעביר אותם ל-"/lib" או ל-"/usr/lib". אם נשתמש באופציה **l-** תוכל התכנית להיות מקושרת לפונקציות בספריה. לדוגמה, נשתמש בפקודה

```
$cc math.c -lkalpha
```

כדי לקשור את הספריה "/usr/lib/libkalpha.a" לקובץ המקור "math.c". שים לב שהאופציה **l-** מורה לפקודה **cc** לחפש ב-"/lib" וב-"/usr/lib" בכדי לאתר את הספריה "libkalpha.a", מכיון שהמלה 'kalpha' נכתבה לאחר האופציה **l-**. זו הסיבה שאם רוצים להשתמש באופציה **l-**, חייבים כל שמות הספריה להתחיל ב-'lib' ולהסתיים ב-'.a'. אין זאת אומרת שאי אפשר לתת כל שם אחר לספריה, אלא שבמקרה זה חייבים לציין את שם המסלול המלא בשורת הפקודה **cc**.

8.7 מדידה ושיפור של ביצועי התכנית

תמיד כדאי למדוד את זמן הביצוע של התכנית לאחר שהוכח שהיא פועלת כראוי, במטרה לשפר את הביצועים שלה. דבר זה הינו שלב חשוב בדרכה של כל מערכת לפני התקנתה הסופית.

לדוגמה, ניתן למדוד את זמן הביצוע של תכנית "math.e" בעזרת פקודת **time** כדלקמן:

```
$time math.e >/dev/null
```

הפלט, לדוגמה:

```
10.2 real 3.2 user 1.1 sys
```

הערך 'real' מצביע על הזמן הכולל שחלף מתחילת הביצוע של התכנית ועד לסיומה. הערך 'user' מצביע על משך הזמן הכולל שהוקדש לביצוע התכנית של המשתמש (ללא קריאות מערכת) והערך 'sys' מצביע על משך הזמן הכולל שהוקדש על ידי המערכת לביצוע קריאות לפונקציות המערכת (ראה בפרק **3** הסבר על ניתוב ל-"/dev/null").

ניתן לקבל מידע מפורט המכיל את חלוקת הזמן בין השגרות השונות של התכנית על ידי שימוש באופציה **p-**. יש להשתמש באופציה זו הן בשלב ההידור והן בשלב הקישור. המהדר מפיק קוד המונה את מספר הקריאות לכל שגרה. תכנית הקישור גורמת ליצירת הקובץ "mon.out" לאחר ביצוע סדיר של התכנית. בעזרת הפקודה **prof** ניתן לסקור את תוכן הקובץ "mon.out" בכדי לקבל את פרופיל הביצוע של התכנית.

לדוגמה, הפקודות הדרושות בכדי להפיק את פרופיל הביצוע של התכנית "math.c" הן:

```
$cc -p math.c
$a.out
$prof a.out
```

פקודת **prof** האחרונה יכולה להפיק את הנתונים הסטטיסטיים המוצגים בטבלה **8.1**.

מטבלה **8.1** ניתן להסיק שרוב הזמן (51.9%) הוקדש לפקודת **read**. אם לדוגמה, התכנית "math.c" תשתמש בפקודת **read** במקום פקודת **getc** כדי לבצע מספר גדול של העברות בתים בודדים, היא תהיה בזבזנית מאוד מבחינת זמן. אם נשתמש בפרופיל הביצוע להסקת מסקנות כאלו נוכל לשפר את ביצועי התכנית במידה רבה.

טבלה 8.1 פרופיל ביצוע של תכנית

Time	Seconds	Cumsecs	#Calls	msec/call	Name
51.9	0.56	0.56	781	0.72	read
14.8	0.16	0.72	153	1.05	write
9.3	0.08	0.82			lodpgm
5.6	0.05	0.88			CQRTIM
3.7	0.04	0.92			tparam
1.9	0.02	0.94			A-XEXT
1.9	0.02	0.96			mcount%
1.9	0.02	0.98	16	1.2	sigtrap%

8.8 הידור ושימוש בקובצי פקודות מעטפת

ניתן לחסוך בזמן ולמנוע שגיאות, אם נשתמש בקובץ פקודות מעטפת לביצוע ההידור. הבה נתבונן במקרה שבו יש להפיק תכנית הנקראת "accounts.e" מהקבצים "pay.c" ו-"recv.c" תוך שימוש באופציות **-lm** ו-**-lcurses**. בנוסף לכך יש להעביר את "accounts.e" למדריך "/usr/john/exec". למשימה זו נשתמש בתכנית עריכה, כדי ליצור קובץ המכיל את השורות הבאות:

```
echo "Compilation begins\n"
echo "cc -o accounts.e pay.c recv.o -lcurses -lm\n"
cc -o accounts.e pay.c recv.o -lcurses -lm
echo "Moving accounts.e to /usr/john/exec\n"
mv accounts.e /user/john/exec
echo "End of shell script comp\n"
```

ניתן לקובץ פקודות המעטפת הזה את השם "comp" ונשמור אותו. כדי ליצור קובץ ביצוע נשתמש בפקודה

```
$chmod +x comp
```

את קובץ הפקודות (התכנית) נוכל לבצע על ידי הקשת השם שלו (ראה גם פרק 11).

פרק 9

פונקציות ספריה סטנדרטיות בשפת C

תכניות צריכות בין השאר, להקצות, לתרגם ולהשוות תווים ומחרוזות. בשפת C קיים מגוון גדול של פונקציות ספריה המטפלות בתווים ומחרוזות. הפונקציות המטפלות במחרוזות נקראות בכל פעם שמבצעים הידור של תכנית C. כדי להשתמש בפונקציות המטפלות בתווים יש לכלול בתכנית את ההוראה הבאה:

```
#include <ctype.h>
```

9.1 פונקציות העוסקות בתווים

ניתן להשתמש בכל אחת מהפונקציות המפורטות בטבלה 9.1 כדי לבדוק תווים. כל הפונקציות מחזירות ערך השונה מאפס אם תוצאת הבדיקה היא אמת ואפס - אם תוצאת הבדיקה היא שקר.

טבלה 9.1 פונקציות ספריה בשפת C המשמשות לבדיקת תווים

הפונקציה	הבדיקה
isdigit(c)	c - ספרה עשרונית בתחום שבין 0 ל-9.
isascii(c)	c - תו ASCII הקטן מ-200 בבסיס 8.
isalpha(c)	c - אות.
islower(c)	c - אות קטנה באנגלית.
isupper(c)	c - אות גדולה באנגלית.
isalnum(c)	c - תו אלפאנומרי (אות או מספר).
isxdigit(c)	c - ספרה בבסיס 16 בתחום שבין 0 ל-9 או אחד התווים שבין a ל-f או בין A ל-F.
isspace(c)	c - רווח, טאבולטור, סוף שורה (CR), הזנת נייר (form-feed), או רווח אנכי.
isprint(c)	c - תו הניתן להדפסה עם ערך שבין 040 לבין 0176 בבסיס 8.
isgraph(c)	זהה ל-isprint(c), אך מחזיר "שקר" (כלומר, אפס במקרה של רווח).
iscntrl(c)	c - תו בקרה עם ערך 0177 בבסיס 8, או ערך הנמצא בין 0 ל-037 בבסיס 8.
ispunct(c)	c - תו פיסוק (לא תו אלפאנומרי ולא תו בקרה).

הקטע הבא מתכנית C קורא מהקלט הסטנדרטי ספרות בתחום שבין 0 ל-9 ולאחר מכן בונה את המספר המתאים:

```
int c,number=0;
while( isdigit( c=getchar() ) )
     number=number * 10 + c - 48;
```

ניתן להשתמש בפונקציות toupper(c) , ו-tolower(c) להמרת "גודל" האותיות (באנגלית כמובן):

* **tolower(c)** ממיר תווים לאותיות קטנות. מחזיר את התו ללא שינוי, אם איננו אות גדולה.
* **toupper(c)** ממיר תווים לאותיות גדולות. מחזיר את התו ללא שינוי, אם איננו אות קטנה.

בקטע התכנית הבאה של תכנית C קוראים תו מהקלט הסטנדרטי. אם התו הוא אות גדולה, ממירים אותו לאות קטנה:

```
int c;
char lw;
c=getchar();
if( isupper(c) )
     lw=tolower(c);
```

9.2 פונקציות המטפלות במחרוזות

פונקציות מחרוזת ב-C יכולות לשמש להשוואה, להעתקה, לשרשור ולמציאת אורך של מחרוזת. ניתן להשתמש במספר פונקציות מחרוזת לטיפול בכל התווים במחרוזת, או במספר תווים מתוך המחרוזת.

השוואת מחרוזות

הפונקציה **strcmp** משווה שתי מחרוזות. המבנה שלה הוא:

```
strcmp(s1,s2)
```

s1 ו-**s2** הינם מצביעים למחרוזות שיש להשוות ביניהן. אם המחרוזות שוות, מחזירה הפונקציה אפס. אם המחרוזות אינן שוות, היא מחזירה את ההפרש בין ערכי ASCII של זוג התווים הראשון שאינו שווה. היא מחסירה תמיד את ערכו של התו במחרוזת השניה מהתו במחרוזת הראשונה.

לדוגמה, הקריאה לפונקציה

```
strcmp("anne","anne")
```

מחזירה את הערך אפס, מכיון שהמחרוזות שוות. אבל, אם נכתוב

```
strcmp("ANNA","ANNE")
```

תחזיר הפונקציה את הערך 4- לאחר שתחסר את התו E מ-A.

הפונקציה **strncmp**, שלא כמו **strcmp**, משווה מספר מסוים של תווים בשתי המחרוזות.

בקטע הבא מתוך תכנית C, משמשת הפונקציה **strncmp** להשוואת תת המחרוזת "Jul" שבמחרוזת "s" עם תת מחרוזות אחדות שבמחרוזת "ms" עד למציאת שם החודש המתאים. שים לב שרק שלושה תווים מושווים בו זמנית:

```
static char *s="Mon Jul 25 12:28:50 MDT 1988";
static char *mn="JanFebMarAprMayJunJulAugSepOctNovDec";
int m;
for(m=1;m<13;m++)
     if( ! strncmp(s+4,mn+3*(m-1),3) )
      break;
printf("Month number=%d\n",m);
```

העתקת מחרוזות

הפונקציה **strcpy** הינה בעלת המבנה הבא:

```
strcpy(s1,s2)
```

פונקציה זו מעתיקה את המחרוזת "s2" לתוך המחרוזת "s1". לדוגמה, קטע התכנית הבא מתוך תכנית C כדי להעתיק את המחרוזת "/usr/anne" למחרוזת "user":

```
static char *name="/usr/anne";
char user[15];
strcpy(user,name);
```

הפונקציה **strncpy**, שלא כמו **strcpy**, מעתיקה רק מספר מסוים של תווים למחרוזת נתונה. בדוגמה הבאה מעתיקה פונקציה זו את ארבעת התווים הראשונים של המחרוזת "name" (כלומר, "/usr") לתוך המחרוזת "user":

```
static char *name="/usr/anne";
char user[15];
strncpy(user,name,4);
user[4]='\0'; /* end string with the null character*/
```

שרשור מחרוזות

הפונקציה **strcat** הינה בעלת המבנה הבא:

```
strcat(s1,s2)
```

פונקציה זו מוסיפה את התווים של המחרוזת "s2" לסופה של המחרוזת "s1". בדוגמה הבאה מתוספת המחרוזת "anne/swap" אל קצה המחרוזת "name", כדי לייצר את המחרוזת "/usr/anne/swap":

```
static char name[20]="/usr/";
strcat(name,"anne/swap");
```

הפונקציה **strncat** מוסיפה תו אחד או יותר לסופה של מחרוזת נתונה. בדוגמה שלהלן, מתוספים ארבעת התווים הראשונים של המחרוזת "user" לסופה של המחרוזת "name", כדי ליצור את המחרוזת "/usr/anne":

```
static char name[20]="/usr/";
static char *user="anne/swap";
strncat(name,user,4);
```

אורך המחרוזת

הפונקציה **strlen** מחזירה את מספר התווים במחרוזת, עד לתו הריק הראשון (אך לא כולל אותו). מבנה הפונקציה:

```
strlen(s)
```

התו s מציין את המצביע למחרוזת. בדוגמה שלהלן, שם החודש מתקבל מהקלט הסטנדרטי. אם שם החודש מכיל יותר מ-3 תווים, תודפס הודעת שגיאה:

```
char month[5];
gets(month);
if( strlen(month) > 3 )
     {
```

```
printf("Only 3 characters allowed for month name\n");
exit(2);
}
```

קריאת מחרוזת

הפונקציה **sscanf** משמשת לקריאת ערכים ממחרוזת, שעשויה להכיל ערכים המיוצגים בדרכים שונות. מבנה הפונקציה:

```
sscanf(s,format,args)
```

* **s** מצביע (pointer) למחרוזת שיש לקרוא.
* **format** צורת הייצוג של הנתונים שיש לקרוא.
* **args** המשתנים שיקבלו את הערכים.

צורות הייצוג הנפוצות ביותר הן:

* **%d** עשרוני
* **%o** בסיס 8 (אוקטלי)
* **%x** בסיס 16 (הקסהדצימלי)
* **%f** נקודה צפה
* **%s** מחרוזת

בדוגמה שלהלן, מתקבלים ומודפסים החודש, השעה והשנה מהמחרוזת "datestr". שים לב לכך ש-'%*s' משמעותו 'התעלם מחלק זה של המחרוזת'.

```
static char *datestr="THU MAR 24 11:04:40 EST 1988";
char month[4];
char year[5];
char ttime[9];
sscanf(datestr,"%*3s%3s%*2s%8s%*3s%4s",month,ttime,year);
printf("month=%s and ttime=%s and year=%s\n",month,ttime,year);
```

כתיבת ערכים שונים למחרוזת

הפונקציה **sprintf** משמשת לכתיבת מספר ערכים למחרוזת נתונה.

מבנה הפונקציה:

```
sprintf(s,format,args)
```

הסבר הפרמטרים:

* **s** מצביע למחרוזת שתקבל את הערכים.
* **format** מצביע למחרוזת המגדירה את הערכים שיש לכתוב למחרוזת s.
* **args** רשימה של משתנים, או ערכים, שיש לכתוב.

בדוגמה שלהלן, כותבים את המחרוזת

```
pr -f /usr/anne/report ¦ lp -n2
```

אל המחרוזת 'out'. הפקודה **system** שבאה לאחר מכן, משמשת להדפסת שני עותקים מקובץ "report":

```
char out[50];
static char *p="/usr/anne/report ";
char *y="| lp -n"
short int d=2;
sprintf(out,"pr -f %s%s%d\n",p,y,d);
system(out);
```

המרת מחרוזת לערך שלם

שתיים מבין הפונקציות הנפוצות ביותר להמרת מחרוזת של ספרות לערך מספרי הן **atoi** ו-**atol**. המבנה שלהן הוא:

```
int atoi(s)   long atol(s)
char *s;      char *s;
```

הפונקציה **atoi** מחזירה ערך שלם המיוצג על ידי מחרוזת התווים ש-s מצביע עליה. הפונקציה **atol** מחזירה ערך שלם ארוך, המיוצג על ידי מחרוזת התווים ש-s מצביע עליה.

הדוגמה שלהלן מראה איך ניתן להמיר מחרוזת של תווים למספר ארוך:

```
static char *s= "9958567";
long atol(),ret;
ret=atol(s);
printf("string was=%s and number= %ld\n",s,ret);
```

ולבסוף, כדאי לציין בנקודה זו שישנן שתי פונקציות נוספות שבעזרתן ניתן לבצע המרה.

הפונקציה האחת היא:

```
long strtol(s,p,base)
    char *s, **p;
    int base;
```

כאשר (int) strtol(s,(char**)NULL,10) מקביל ל-**atoi(s)**.

הפונקציה השניה היא:

```
double atof(s)
char *s;
```

פונקציה זו ממירה את המחרוזת ש-**s** מצביע עליה למספר מקביל המיוצג בשיטת הנקודה הצפה בדיוק כפול (double-precision floating-point number).



פרק 10

מאפייני תקשורת בין תהליכים פנימיים ופקודות מערכת

בפרק זה נדון במנגנונים המסורתיים והחדשים שמאפשרים תקשורת בין תהליכים פנימיים (להלן, IPC - InterProcess Communication). קבצים, אותות, צינורות ללא שם וצינורות בעלי שם הינם אמצעים מסורתיים שמאפשרים תקשורת בין תהליכים פנימיים. אמצעי IPC החדשים הינם בעלי עוצמה רבה וכוללים תורי הודעות, סגמנטים משותפים בזיכרון ואתתים. בפרק זה גם נציג מספר פקודות מערכת שהינן חיוניות לפיתוח מוצרים ב-UNIX. מקובל לכנות את פקודות המערכת גם בשם "פונקציות".

10.1 הפקודה system

הפקודה **system** מאפשרת לקרוא ולבצע תכנית מתוך תכנית אחרת ולהחזיר את השליטה לתכנית המקורית.

```
system(command)
```

הפרמטר **command** הינו מצביע למחרוזת המכילה את שורת הפקודה של המעטפת. לדוגמה, משפט בתכנית C:

```
system("date");
```

פקודה זו תגרום לכך שהתאריך והשעה יודפסו בפלט הסטנדרטי. באופן דומה נוכל לכתוב

```
system("pr -f temp.c ¦ lp");
```

הפקודה תגרום לכך שהקובץ "temp.c" יודפס מתוך תכנית C. קריאת המערכת מחזירה את הערך 1- במקרה של שגיאה.

10.2 ההוראה exit

הפונקציה **exit** גורמת להפסקת תכנית אשר רצה ולהחזרת השליטה למערכת ההפעלה.

```
exit(status)
```

הפרמטר **status** הינו ערך שלם שיישלח למערכת. הדוגמה הבאה,

הכתובה בשפת **C**, מדגימה את השימוש בפונקציה **exit**:

```
if ( ( fp=fopen("temp","r") ) == NULL)
     {
     print("Cannot open file temp for reading\n");
     exit(1);
     }
exit(0);
```

10.3 פונקציית המערכת exec

הפונקציה **exec** גורמת להסבת התהליך הקורא לתהליך חדש. למעשה, הפקודות והנתונים של התהליך שקרא מוחלפים על-ידי הפקודות והנתונים של התהליך שצויין כארגומנט בזמן הקריאה ל-**exec**. הפונקציה **exec** תחזיר את השליטה לתהליך שקרא לה, רק אם קרתה שגיאה בזמן הקריאה (לדוגמה, לא ניתן לאתר את התכנית שקראה לפונקציה). ניתן לומר שהקריאה ל-**exec** דומה יותר ל-**goto** מאשר קריאה לפונקציה.

ישנן מספר צורות של קריאה לפונקציה **exec** שחלקן מוצגות להלן:

```
int execl(path,arg-0, arg-1,....arg-n, 0)
char *path, *arg-0, *arg-1,....*arg-n;

int execv(path,argv)
char *path, *argv[];

int execlp(filename,arg-0, arg-1,....arg-n, 0)
char *filename, *arg-0, *arg-1,.... *arg-n;
```

דוגמאות

1. נתון המשפט הבא בשפת C

```
execl("/bin/date", "date", 0);
```

הוא יבצע את הפקודה **/bin/date**. הנוהג הוא כי "arg-0" הינו שם התכנית שיש לבצע, אשר בדוגמה זו היא תקבל את הערך 'date'. הארגומנט האחרון צריך להיות שווה לאפס.

2. אם לא צויין שם מסלול החיפוש בזמן הביצוע של הפקודה **execlp**, היא תשתמש במסלול הנמצא במשתנה הסביבה PATH.

לדוגמה, נתון המשפט הבא בשפת C

```
execlp("ls", "ls", "file-a", "file-b", 0);
```

הוא יבצע את הפקודה **/bin/ls**, כדי לבדוק אם הקבצים "file-a" ו-"file-b" קיימים במדריך הנוכחי.

3. פקודת **execv** הנראית בתכנית C שלהלן משמשת בדרך כלל להעברת ארגומנטים לתכנית אחרת, כאשר המספר המדויק של הארגומנטים אינו ידוע מראש:

```
#include <stdio.h>
main (argc,argv)
int argc;
char *argv[];
{
int i;
char *cmd[20];
cmd[0]="ls"; /* first store the ls command */
/* now store all arguments passed to this program */
for (i=1; ( i < argc ) && ( i < 19 ); i++)
     cmd[i] = argv[i];
cmd[i] = 0;
/* execute the ls command with all the arguments */
execv ("/bin/ls", cmd);
printf ("Error: Cannot execv the 'ls' command \n");
exit(1);
}
```

10.4 פקודות exec ותווים שמורים

החיסרון העיקרי של פקודות **execl** ו-**execv** הינו באי יכולתן להשתמש בתווים שמורים כמו *,?,[,],< ו->. ניתן להתגבר על בעיה זו על ידי כך שנשתמש ב-**execl** כדי לקרוא למעטפת ולאחר מכן נאפשר למעטפת לבצע את הפקודה. קריאה כזו נראית כך:

```
execl("/bin/sh", "sh", "-c", command, 0)
```

לדוגמה, כדי לקבל רשימה של כל הקבצים בעלי הסיומת ".c" נשתמש בפקודה הבאה:

```
execl("/bin/sh", "sh", "-c", "ls *.c", 0)
```

10.5 הפונקציות fork ו-wait

פקודות **exec** אינן שימושיות במיוחד, מכיון שהן מספקות מנגנון שבו התכנית הקוראת מסיימת את פעולתה והתכנית שנקראה ממשיכה להתבצע.

הפונקציה **fork** מאפשרת ליצור מתוך תכנית שמתבצעת שני תהליכים עצמאיים, אשר יכולים לפעול בנפרד. לאחר הקריאה ל-**fork** נוצר תהליך חדש, **תהליך בן**. זהו העתק מדויק של התהליך שקרא לפונקציה, אשר נכנה אותו **תהליך אב**. תבנית הקריאה לפונקציה:

```
pid=fork()
```

לאחר הקריאה ל-**fork**, ניתן לזהות את תהליך האב ואת תהליך הבן על ידי בדיקת הערך המוחזר. אם חזר הערך "אפס" הרי שזהו תהליך בן. אם חזר מספר שלם חיובי, אשר הוא למעשה המספר המזהה של התהליך, לפנינו תהליך האב. ולבסוף, אם חזר הערך 1- פירוש הדבר שהקריאה ל-**fork** נכשלה.

קריאה לפונקציה **wait** גורמת לתהליך האב (שקרא לפונקציה) להמתין עד אשר יסיימו התהליכים הבנים לפעול, ורק אז הוא יוכל לחדש את פעולתו שלו. הקריאה ל-wait נראית כך:

```
wait(pint)
```

pint הינו מצביע למשתנה המקבל מספרים שלמים. התכנית שלהלן מדגימה שילוב של קריאות ל-**fork** ול-**exec**. התכנית שקוראת לפונקציה מבצעת את התכנית "master" הנמצאת במדריך "/usr/andreas". לאחר שהתכנית "master" מסתיימת, המערכת מחזירה את השליטה לתכנית שקראה לפונקציה.

```
#include <stdio.h>
main()
{
int status, pid;
pid=fork();
if( pid == 0 )
    {
    /* its the child process * /
    execl ("/usr/andreas/master", "master", NULL);
    printf ("Error in exec of master program\n");exit(1);
    }
else if( pid == -1 )
```

```
    printf("Failed to fork\n");
    exit(1);
     }
else
   /*its parent process so wait for child process to finish */
    wait(&status);
printf("End of program\n");
exit(0);
}
```

הסטטוס המוחזר על ידי הפונקציה **wait** (בגרסת UNIX-V) מיוצג על ידי מספר שלם שנוצר מ-2 קבוצות של שמונה סיביות. שמונה הסיביות הנמוכות מכילות את הערכת המערכת לגבי סטטוס הסיום של תהליך הבן. הסטטוס יהיה "אפס" לסיום נורמלי ושונה מ"אפס" כדי לייצג תקלות שונות כמו סיום כתוצאה מפסק, או כתוצאה מאות סיום.

שמונה הסיביות הגבוהות נלקחות מארגומנט הקריאה ל-**exit** של תהליך הבן לפני שהוא מסתיים. אם תהליך הבן לא קרא באופן מפורש לפונקציה **exit**, יהיה הסטטוס של שמונה הסיביות הגבוהות שווה ל-0.

לסיום נוסיף, כי תהליך הבן יורש מאפיינים רבים מתהליך האב כמו: הסביבה, הקבצים הפתוחים, הקלט הסטנדרטי, קבצי השגיאה והפלט, ברירות המחדל בזמן יצירת קובץ, טיפול באותות וסגמנטים משותפים בזיכרון.

10.6 אותות

אותות (signals) מיועדים באופן מסורתי כדי לסיים את פעולתם של תהליכי UNIX. אותות יכולים להשתמש בערכים מוגדרים מראש בלבד וכתוצאה מכך הם אינם מיועדים להחלפת מידע.

אות הינו צורה של פסק תוכנה (software interrupt), אשר מפסיק את הביצוע הנורמלי של תכנית. פסק יישלח לתהליך אשר רץ, אם התרחשו הדברים הבאים:

* טעויות בחישובי נקודה צפה.
* הפניות לא חוקיות בזיכרון.
* ביצוע של הוראה לא חוקית.
* הקשת DEL.
* הקשת QUIT.
* שימוש בפקודת המערכת **kill**.
* קריאה לשגרות מערכת (כמו "abort", אשר מפיקה את האות SIGIOT, או "alarm", אשר מפיקה את האות SIGALARM).

לאחר שהופק אות, עדיין ניתן במספר מקרים למנוע את התוצאה הסופית, שהיא הפסקת התהליך. במקרים אלה ניתן ליירט את האות ולהתעלם ממנו, או לנקוט בפעולה אחרת.

פונקצית המערכת kill

הפונקציה **kill**, המשמשת להפקת אות:

```
int kill(pid,sig)
int pid,sig;
```

אם המספר המזהה (ID) של המשתמש בתהליך שקרא ל-kill אינו של משתמש-על, תהיה פקודת **kill** אפקטיבית רק כאשר המספר המזהה של המשתמש בתהליך שקרא זהה למספר המזהה של המשתמש בתהליך המקבל. הפרמטר PID יכול לקבל מספר ערכים:

PID>0	שלח "sig" לתהליך שהמספר המזהה שלו הינו PID (Process ID).
PID=0	שלח "sig" לכל התהליכים (פרט לתהליך 0 ותהליך 1), שבהם המספר המזהה של קבוצת התהליך שווה לזה של השולח.
PID=-1	שלח "sig" לכל התהליכים, שבהם המספר המזהה האמיתי של המשתמש זהה למספר המזהה האפקטיבי של המשתמש בתהליך השולח. אם "sig" נשלח על ידי משתמש על, אזי כל התהליכים במערכת (פרט ל-0 ו-1) יקבלו את האות.
PID<-1	שלח "sig" לכל התהליכים, אשר המספר המזהה של הקבוצה שלהם שווה לערך האבסולוטי של PID.

לדוגמה, בתכנית בשפת C ישנו המשפט הבא:

```
kill (getpid(),SIGTERM)
```

הפונקציה תשלח לתהליך שקרא לפונקציה אות סיום שניתן "ללכוד" אותו.

לעומתו, הפונקציה במשפט

```
kill (152,SIGKILL)
```

תשלח לתהליך מספר 152 אות סיום שלא ניתן "ללכידה", והוא יסתיים מייד ללא תנאי.

אם הקריאה ל-**kill** עברה בהצלחה, יהיה הערך המוחזר שווה לאפס. בכל מקרה אחר יהיה הערך המוחזר שווה 1-.

פונקצית המערכת signal

קוראים לפונקצית המערכת **signal** כאשר יש צורך בעיבוד אותות.
הקריאה לפונקציה:

```
signal(signalno,action)
```

כאשר

signalno מספר שמציין את מספר האות.
action מציין את הפונקציה שיש לבצע כאשר האות מתקבל. הוא גם יכול לציין את הערכים SIG_IGN, שמשמעותו התעלמות מהאותות ואת SIG_DFL, שמאפשר שיחזור של ברירת המחדל לסיום התהליך.

כל מספרי האותות יכולים להיות זמינים לתכנית C, אם נכתוב את המשפט הבא:

```
#include <signal.h>
```

דוגמאות:

* כדי להתעלם מהפסק הנוצר בעת הקשה על DEL השתמש במשפט הבא:

```
signal (SIGINT,SIG_IGN)
```

* כדי להתעלם מהאות QUIT השתמש במשפט הבא

```
signal (SIGQUIT,SIG_IGN)
```

* כדי לשחזר את ברירת המחדל במקרה של קבלת האות QUIT השתמש במשפט הבא:

```
signal (SIGQUIT,SIG_DFL)
```

בעת הקריאה ל-**signal**, אם שדה **action** מכיל קריאה לפונקציה, יש להכריז עליה לפני שמשתמשים בה. בצורה זו יוכל המהדר להתמודד עם ההפניה שבהמשך.

כאשר מתקבל אות ולא מתעלמים ממנו (פרט ל-SIGILL ו-SIGTRAP),

הוא חוזר באופן אוטומטי לערך המחדל שלו. אי לכך, תכנית הרוצה לטפל במספר פסקים, צריכה לקרוא לשגרת **signal** בכל פעם שמתרחש פסק. בתכנית "alarm.c" תוכל למצוא הדגמה קצרה של הדרך שבה יש להשתמש בקריאה ל-**signal**.

10.7 השעיית הביצוע של תהליך

ההוראה **pause** יכולה לשמש להשעיית הביצוע של התהליך הקורא, עד אשר התהליך מקבל אות שאין הוא יכול להתעלם ממש.

```
pause()
```

אם הפונקציה לתפיסת האות אינה מסיימת את התהליך הקורא, יחדש התהליך את פעולתו מאותה נקודה שבה הושעה.

10.8 הפעלה של פסקי זמן

לעתים רוצים שתכנית תפסיק את פעולתה אם אירוע מסוים לא התרחש תוך פרק זמן מוגדר מראש. ניתן ליצור פסק זמן כזה על ידי קריאה לפונקצית המערכת **alarm** ובעזרת האות "SIGALARM". הקריאה ל-alarm:

```
unsigned alarm(secs)
unsigned secs;
```

הפרמטר **secs** משמש לציון מספר השניות שיחלפו לפני שהאות SIGALARM יישלח לתהליך שקרא לפונקציה. קריאה ל-**alarm** עם הערך אפס תבטל את בקשת **alarm** הקודמת.

התכנית שלהלן מדגימה קריאה ל-**alarm**. אם המשתמש לא יקיש דבר במשך 10 שניות, התכנית תיקח פסק זמן ולאחר חמשה פסקי זמן רצופים היא תפסיק את פעולתה.

```
/*
* Program     : alarm.c
*
* Purpose     : To demonstrate the use of alarm signals in
*               implementing timeouts
**/

#include <stdio.h>
#include <signal.h>
main()
{
```

```
char buf[50];
int count=0;
int alarmdemo();

signal (SIGALRM,alarmdemo);
while( count < 5 )
        {
        alarm(10);  /* set a timeout of 10 seconds */
        buf[0]='\0';
        printf("\nEnter string within 10 seconds > ");
        gets(buf);
        alarm(0);  /* cancel the timeout */
        /* check if something has been read in 10 seconds */
        if( strlen(buf) == 0 ) /* nothing was entered */
            {
            printf("\nTimeout occurred");
            count++;
            }
        else
            {
            printf("Input read within 10 seconds is=%s\n",buf);
            count=5; /* to exit the loop */
            }
        } /* end of while loop */
} /* end of main program */

alarmdemo()
{
signal(SIGALRM,alarmdemo);
printf("\nInside alarmdemo routine");
}
```

הרצה טיפוסית של התכנית תפיק את הפלט הבא:

```
Enter string within 10 seconds >
Inside alarmdemo routine
Timeout occurred
Enter string within 10 seconds > anne
Input read within 10 seconds is=anne
```

10.9 חידוש של פעולת התכנית

ניתן להשתמש בשגרות הספריה **setjmp** ו-**longjmp** כדי קבוע נקודה בתכנית שממנה תחדש התכנית את פעולתה לכשיופיע פסק. השגרות **setjmp** ו-**longjmp** מתנהגות בצורה דומה מאד לפקודות **goto** בשפת **C**, אך ללא ההגבלות הרגילות. המבנה של שגרות אלו הוא כדלקמן:

```
#include <setjmp.h>

int setjmp(svstack)
jmp_buf svstack;

int longjmp(svstack,retval)
jmp_buf svstack;int retval;
```

השגרה הראשונה הינה **setjmp**, אשר שומרת את סביבת המחסנית של התכנית שרצה ומחזירה ערך השווה לאפס. ניתן לקרוא לאחר מכן לשגרת **longjmp** כדי שתשחזר את סביבת המחסנית של התכנית שרצה, לאחר שהביצוע של התכנית הופסק על ידי המשתמש במכוון או בשגגה. שגרת **longjmp** מחזירה את הערך המצוין על ידי הפרמטר **retval**. ניתן לבדוק ערך זה בנקודה שממנה נקראה השגרה **setjmp**. תכנית C שלהלן מראה כיצד ניתן להשתמש בשגרות **setjmp** ו-**longjmp**:

```
#include <setjmp.h>
#include <signal.h>
extern anintr();
jmb_buf svstack;
main()
{
int loadmath,sqroot;
signal(SIGINT, anintr);
/* call setjmp to save the current stack environment from
   this point on*/
setjmp(svstack);
          /*....
          ..... rest of program     ...
          .....  */
exit(0);
```

```
} /*end of main program*/

anintr()
{
/* reset signal for next interrupt occurence */
signal(SIGINT,anintr);
printf("Please do not interrupt program !!\n")
/* restore stack environment after a DEL key stroke */
longjmp(svstack,1);
}
```

10.10 צינורות ללא שם

עד עתה הראינו כיצד ניתן להשתמש בקריאות לפונקציות **fork** ו-**exec** כדי ליצור תהליכי בן. לא הראינו עדיין כיצד יכול תהליך בן להתקשר עם תהליך האב שלו ועם תהליכים אחרים. זאת ניתן לבצע בעזרת מנגנון הצינורות.

צינור (pipe) הוא סוג מיוחד של קובץ, אשר נוצר על ידי תהליך כדי להעביר באמצעותו מידע לתהליך אחר. שלא כמו קובץ רגיל, מייצג צינור שטח אחסון זמני בזיכרון שנשלט כולו על ידי המערכת ואינו קשור לזיכרון של התהליך שיצר אותו. אולם, לתהליכים הכותבים אל הצינור או קוראים ממנו, חייב להיות מקור משותף.

צינורות מכתיבים שימוש במנגנון FIFO (ראשון-נכנס-ראשון-יוצא) בעת איחזור נתונים ופועלים בצורה סדרתית; לא ניתן לדוגמה לבצע חיפוש בצינור. כל הודעה בצינור יכולה להקרא פעם אחת בלבד. צינורות מיועדים לשימוש בין שני תהליכים המשתפים פעולה, בדיוק כמו סימן הצינור של המעטפת, '|', המשמש להעברת הפלט של תהליך אחד לקלט של תהליך אחר.

10.11 פונקציות הצינור

בספריה הסטנדרטית ניתן למצוא מספר פונקציות צינור כמו:

* popen
* pclose
* pipe
* close

ניתן להשתמש בפונקציות אלו לטיפול בצינור (צינור ללא שם) בדיוק כמו שמטפלים בקובץ. הן מחזירות מצביעים לקובץ או מתארי קובץ, אשר יכולים לשמש פונקציות קלט/פלט סטנדרטיות לקריאה מצינור או לכתיבה אליו (לדוגמה: **read**, **write**, **fscanf**, **fprintf** וכד').

popen ו-pclose

הפונקציה **popen** הינה בעלת המבנה הבא

```
popen(cmd, type);
```

הפרמטר **cmd** הינו מצביע למחרוזת שמכילה את שורת הפקודה במעטפת ו-**type** מציין האם יש לפתוח את הצינור לקריאה (r) או לכתיבה (w).

הפונקציה **popen** יוצרת תהליך חדש ולאחר מכן פותחת צינור לקלט או לפלט הסטנדרטי של התהליך החדש. אם הקריאה ל-**popen** מצליחה, מוחזר מצביע לצינור הפתוח הזהה למצביע לקובץ. אם הקריאה לא הצליחה כתוצאה משגיאה כלשהי, אחרת יוחזר הערך NULL.

הפונקציה **pclose** הינה בעלת המבנה הבא

```
pclose(ptr);
```

הפרמטר **ptr** הינו המצביע לצינור שיש לסגור. הדוגמה שלהלן מראה את השימוש ב-**popen** וב-**pclose**.

```
/* Program  : ttyname.c
*  Purpose  : To get terminal name using a pipe
**/
#include <stdio.h>
main()
{
FILE *pfp, *popen();
char buf[12];
if ( ( pfp = popen("tty","r" ) ) == NULL )
    {
   printf("tty pipe call failed\n");
   exit(1);
    }
```

```
fscanf(pfp, "%s", buf);  /* read terminal name from pipe */
printf("ttyname read = %s\n", buf);
pclose(pfp);
}
```

close-ו pipe

פונקצית המערכת **pipe** הינה בעלת המבנה הבא

```
pipe(pfd)
```

הפרמר **pfd** הינו מצביע למערך בן שני איברים של מספרים שלמים. קריאה לצינורות ברמה נמוכה זו פותחת קובץ הן לקריאה והן לכתיבה. היא תחזיר את הערך אפס אם הקריאה הצליחה ואת הערך 1- אם הקריאה לא הצליחה כתוצאה משגיאה כלשהי. כדוגמה, הבה נתבונן בקטע הבא מתכנית C:

```
int pfd[2], rt;
rt = pipe(pfd);
if ( rt == -1 )
     exit(1); /*because an error occurred */
```

האיבר הראשון במערך "pfd[0]" יכיל את מתאר הקובץ לקריאה לצינור, בעוד ש-"pfd[1]" יכיל את מתאר הקובץ לכתיבה לצינור. בסעיף הבא תוכל לראות דוגמה של תכנית C המשתמשת בקריאות לפונקציות המערכת שתוארו בסעיף זה.

10.12 שימוש בצינורות ברמה נמוכה

הפונקציות **read** ו-**write** (בדומה לק/פ ברמה נמוכה) יכולות לשמש לקריאת תווים מצינור ולכתיבת תווים לצינור.

למערכת יש שליטה מלאה על אחסון הנתונים בצינור. אם תהליך כותב לצינור מלא, הוא יצטרך להמתין עד אשר הצינור יתרוקן. אם תהליך קורא מצינור ריק, הוא יצטרך להמתין עד אשר יגיעו הנתונים.

כדי לקרוא 20 תווים מצינור פתוח, אפשר להשתמש בדוגמה הבאה:

```
int pfd[2], count;char buf[20];
count=read( pfd[0], buf, sizeof(buf));
```

כדי לכתוב 20 תווים לצינור פתוח, אפשר להשתמש בדוגמה הבאה:

```
count=write( pfd[1], buf, 20)
```

מכיוון שיש שני מתארי קובץ הקשורים לכל צינור ברמה נמוכה, יש צורך להשתמש בפונקציה **close** פעמיים. להפסקת הקריאה מהצינור נכתוב:

```
close( pfd[0] );
```

להפסקת הכתיבה לצינור נכתוב:

```
close( pfd[1] );
```

התכנית בשפת C שלהלן מדגימה את השימוש בצינורות ואת הקריאות לפונקציות המערכת **fork** ו-**exec**:

```
/* Program    : pipe.c
 *
 * Purpose    : To demonstrate the use of pipes.  It first
                accepts a shell command from the standard
                input. It then passes the command to the
                function "op_pipe" which creates the
                pipe. It then forks and execs the command
                after arranging that the output of the command
                goes to the pipe that was created.  The
                function "op_pipe" then returns a file
                descriptor to the read side of the pipe to
                the main program, which is used to read the
                results put in the pipe by the forked
                execed child process. Thus the parent
                process reads from the pipe.  Hence an
                inter-process communication is achieved
                between the two processes.
                */

#include <stdio.h>
#include <fcntl.h>
#define ERROR (-1)
#define READ 0
#define WRITE 1
main()
{
```

```
int rfd, count;
char cmd[20], c;

printf("Please enter command : ");
gets(cmd);

/* create pipe and write to it; get read file descriptor */
rfd=op_pipe(cmd);
if (rfd < 0 )
    {
    printf("Error in op_pipe function and rfd=%d\n",rfd); .
    exit(1);
    }

/* read the pipe and output its contents until its empty */
while ( ( count=read(rfd, &c, 1) ) != NULL )
    putchar(c);

exit(0);
}       /* end of main program */

/*
* Function      : op_pipe
*
* Purpose       : To create the pipe, redirect standard output
*                 to the pipe, and to write to the pipe by
*                 executing the command that was passed as
*                 an argument. It subsequently returns the
*                 file descriptor to the read side of the
*                 pipe.
**/

static int op_pipe(cmd)
char *cmd;
{
int status, pfd[2], pid;
/*  pfd is file descriptor for pipe
    pfd[READ] for reading the pipe
    pfd[WRITE] for writing to the pipe */
if( pipe[pfd]==ERROR )  /* open pipe */
   return(-1); /* failed to open file */
```

```
if( ( pid=fork() )==0 )
   {
   /*child process code */
   /* close stdout ;see fcntl command below */
   close(stdout);
   close(pfd[READ]);      /* close read side of pipe. */
   /*use fcntl so that stdout refers to the same open pipe as
   pfd[WRITE] */
   fcntl(pfd[WRITE], F_DUPFD, stdout);
   /* close old pfd [WRITE] since it is no longer needed */
   close(pfd[WRITE]);
     /* execute the command that was given by the user */
     /* output will now be written to the pipe */
   execlp("sh", "sh", "-c", cmd, NULL);
   printf("Error during exec\n");
   return(-3);
   }
else if( pid==ERROR )
   {
   printf("Failed to fork\n");
   return(-4);
   }

/* parent code */
wait( &status ); /* wait for child to die */
close(pfd[WRITE] );
return(pfd[READ] );
}       /* end of function */
```

10.13 צינורות בעלי שם

פעולת צינורות בעלי שם זהה לזו של צינורות ללא שם. אולם לתהליכים המשתמשים בצינורות בעלי שם לא חייב להיות מקור משותף. תהליכים שאינם קשורים ביניהם יכולים לשלוח בקשות לצינור בעל שם, בתנאי שהם יודעים את שמו. ניתן ליצור צינורות בעלי שם מרמת הפקודה ב-UNIX בעזרת הפקודה **mknod**:

```
$/etc/mknod(s) pname p
```

הפרמטר **p** מציין שרוצים ליצור צינור הנקרא "pname",

ולא ליצור קובץ מיוחד, שהוא השימוש הנפוץ בפקודת **mknod**.

גם מתוך תכנית C ניתן ליצור צינור בעל שם בעזרת הפקודה **mknod**. לדוגמה, שימוש בפקודה **mknod**:

```
#include <fcntl.h>
char buf[512], *pname="mypipe"; int fdpipe;
mknod(pname, 010000 | 0664,0);
```

פקודה זו תיצור צינור בעל שם הנקרא "**mypipe**" ובעל הרשאת גישה 0664.

לאחר מכן ניתן לפתוח את הצינור לקריאה באמצעות פונקצית המערכת **open**, כפי שנראה בדוגמה שלהלן:

```
fdpipe=open(pname,O_RDONLY | O_NDELAY);
```

ולבסוף, כדי לקרוא יש לפנות לפונקצית המערכת **read**

```
read(fdpipe,buf,sizeof(buf));
```

צינור בעל שם ימשיך להתקיים עד אשר יבוטל בעזרת קריאה לפונקצית המערכת **unlink**.

10.14 דו"ח מצב של אמצעי IPC

פקודת **ipcs** משמשת לדווח על מצב אמצעי IPC (סגמנטים משותפים בזיכרון, אתתים ותורי הודעות). התחביר של הפקודה:

```
ipcs [אופציות]
```

אם לא צוינו אופציות כלשהן, **ipcs** תדווח על המצב של אמצעי ה-IPC. להלן כמה מבין האופציות הנפוצות ביותר:

-m הצג מידע על הסגמנטים המשותפים שפעילים כרגע בזיכרון.
-q הצג מידע על תורי ההודעות הפעילים.
-s הצג מידע על האתתים הפעילים.
-a דווח כל מה שאפשר על אמצעי IPC.

10.15 ביטול מידע על אמצעי IPC

פקודת **ipcrm** משמשת לביטול מבני נתונים ומספר זיהוי הקשורים לאמצעי IPC מסוים. התחביר של הפקודה הוא

```
ipcrm [אופציות]
```

להלן כמה מבין האופציות הנפוצות ביותר:

m shmid- מבטל סגמנט משותף בזיכרון הקשור למזהה **shmid**.
s semid- מבטל מערכת של אתת הקשורה למזהה **semid**.
q msqid- מבטל תור הודעות הקשור למזהה **msqid**.

10.16 תורי הודעות

תור הודעות (Message Queues - MQ) הינו שטח בזיכרון הראשי אשר מופרד מהזיכרון של התהליך שיצר אותו. לאחר שנוצר, יכולים מספר תהליכים להתקשר ביניהם על ידי שיגור הודעות לתור וקבלת הודעות מהתור.

כל הודעה בתור ההודעות חייבת להיות מסווגת (type). בדרך זו יש לתהליך המקבל אפשרות לבחור בסוג הודעה מסוים. בדומה לצינורות, מועברות ההודעות לפי שיטת FIFO. כל כניסה לתור ההודעות היא אטומית (לא ניתן לבצע שתי פעולות בו-זמנית לתור, פעולה אחת חייבת להסתיים לפני שהפעולה הבאה מתחילה). כל תהליך שיש לו הרשאה מתאימה יוכל להכנס לתור ההודעות. תורי ההודעות מתאימים ביותר להחלפת הודעות קצרות בין תהליכים.

תור ההודעות יתקיים כל עוד לא בוצעה קריאה לפונקצית המערכת **msgctl**, או לפונקצית המערכת **ipcrm -q** (ראה גם הסברים על פונקציות המערכת **ipcs**, **ipcrm** ו-**msgctl**).

תכניות המטפלות בתורי הודעות יכילו באופן רגיל שלושה קבצי כותרת:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

ipc.h הינו קובץ המשמש את התקשורת בין תהליכים פנימיים, ותפקידו לציין את הרשאות הגישה. הקובץ מכיל את הגדרות הבאות:

IPC_ALLOC, IPC_CREATE, IPC_NOWAIT, IPC_EXCL, IPC_SET, IPC_STAT, IPC_RMID ו-IPC_PRIVATE.

בנוסף, הקובץ מכיל את המבנה **ipc_perm**, כפי שמפורט להלן:

```
ipc_perm{
        signed   short uid;     /* real user id of owner*/
        unsigned short gid;     /* real group id of owner*/
        unsigned short cuid;    /* effective user id*/
        unsigned short cgid;    /* effective group id*/
        unsigned short mode;    /* access modes*/
        unsigned short seq;     /* slot usage sequence number*/
        long     key };         /* the key*/
```

הקובץ המשמש את תור ההודעות, "msg.h", מכיל מספר מבנים, שאחד מהם הינו מבנה הנתונים "msqid_ds", המכיל את המספר המזהה של תור ההודעות. מבנה הנתונים "msqid_ds" קיים לכל תור הודעות במערכת והמבנה שלו נראה להלן:

```
msqid_ds{
     struct ipc_perm msq_perm;/*the operations permission
                              structure*/
     struct msg *msg_first;/*pointer to first message in queue*/
     struct msg *msg_last;/*pointer to last message in queue*/
     unsigned short msg_cbytes;/*number of bytes currently in the
                              queue*/
     unsigned short msg_qnum;/*number of messages in queue*/
     unsigned short msg_qbytes;/*maximum number of bytes in
                              queue*/
     unsigned short msg_lsipd;/*pid of last "msgsnd" call*/
     unsigned short msg_lrpid;/*pid of last "msgrcv" call*/
     long     msg_stime;/*time of last "msgsnd" call*/
     long     msg_rtime;/*time of last msgrcv" call*/
     long     msg_ctime;/*last time of change*/
    };
```

פונקציות מערכת אחדות משמשות ליצירה ולטיפול בתורי הודעות. התחביר שלהן נראה להלן:

```
int msgget(key,flag)
key_t key;
int flag;
```

```
int msgsnd(msqid,msgp,msgsz,flag)
int msqid;
struct msgbuf *msgp;
int msgsz,flag;

int msgrcv(msqid,msgp,msgsz,msgtyp,flag)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int flag;

int msgctl(msqid,cmd,buf)
int msqid,cmd;
struct msqid_ds *buf;
```

הפונקציה **msgget** יוצרת את תור ההודעות יחד עם המבנה הקשור אליו (ראה את הדיון הקודם בעניין "msqid_ds") ומחזירה מספר שלם חיובי שהינו ייחודי ואשר ידוע כמספר המזהה של תור ההודעות ("msqid"). "msqid" יכול לאחר מכן להיות בשימוש של **msgsnd** ו-**msgrcv**. בעזרת קריאה ל-**msgsnd** ניתן לשלוח הודעה לתור הודעות הקשור ל-"msqid". בעזרת קריאה ל-**msgrcv** ניתן לקרוא הודעה מתור ההודעות הקשור ל-"msqid".

בעזרת קריאה ל-**msgctl** ניתן לבצע מספר פעולות בקרה בתור ההודעות. לדוגמה, הפקודה **msgctl(msqid,IPC_RMID)** מבטלת את תור ההודעות הקשור ל-"msqid". בספר ההדרכה של UNIX תמצא תיאור מפורט יותר של הפרמטרים המועברים לפונקציות שתוארו לעיל. כדי להקל את השימוש בפונקציות מערכת אלו, הבאנו את התכנית הבאה. התכנית יוצרת תור הודעות, כותבת לתוכו שתי הודעות, אשר מסווגות כסוג 1 וסוג 2, ואז קוראת בצורה סלקטיבית מהתור את ההודעה מסוג 2 ומדפיסה אותה.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
#define READM 0644
#define ERROR (-1)
struct ipc_perm coname;
struct message   {
              long type;
              char compname[40];
              };
struct message mesg;
extern int errno;
int msqid;
char buf[50];
main(argc,argv)
int argc;
char *argv[];
{
long mtype1=1, mtype2=2;
void in_ms(); /* routine used to initialize structure */
coname.key= (long) getpid();  /* create a unique key */

/* create message queue */
msqid=msgget(coname.key,(IPC_CREAT| IPC_EXCL| READM) );
if( msqid == ERROR )
        {
        printf("Error in creating message queue\n");
        printf("rqid=%d and errno=%d\n",msqid,errno);
        perror(buf);
        exit(1);
        }
system("ipcs -q");   /* show message queue that was created */
/* initialize structure to some data */
in_ms("Corporation ABC",mtype1);
if( wr_mq (msqid,&mesg) ) /* write to message queue */
     exit(1); /* because of an error */
/* initialize structure to another string */
in_ms ("Corporation UNIXA",mtype2);
if( wr_mq (msqid,&mesg) ) /* write to message queue */
    exit(1);
if( rd_mq(msqid,mtype2) ) /* read message queue */
    exit(1);
exit(0);
}
```

```
/*  Function    : To initialize the structure to some data */

static void in_ms(s,thetype)
char *s;
int thetype;
{
mesg.type=thetype;
strcpy(mesg.compname,s);
strcpy(mesg.compname,s);
}

/* Function     : To write a message into the message queue */

static int wr_mq(msqid,pmesg)
int msqid;
struct message *pmesg;
{
if( msgsnd(msqid,pmesg,sizeof(struct message),0 ) == ERROR )
        {
        printf("Error in writing to message queue\n");
        perror(buf);
        return(1);
        }
system("ipcs -q");  /* show message queue */
return(0); /* message was written without problems */
}

/*  Function    : to read message queue */

static int rd_mq(msqid,thetype)
int msqid;
long thetype;
{
int temp;
struct message rmesg,*prmesg;
prmesg= &rmesg;
if( msgrcv(msqid,prmesg,sizeof( struct message),thetype,1)
    == ERROR )
        {
```

```
        printf("Error in reading message type=%d from the
        queue\n",thetype);
        perror(buf);
        return(1);
        }
/* message queue was read successfully */
printf("\nCompany name read=%s\n",prmesg->compname);
return(0);
}
```

ביצוע של התכנית לעיל יפיק את הפלט הבא:

```
 status from /dev/knem as of Mon Dec 19 09:32:53 1988
 T       ID       KEY           MODE          OWNER     GROUP
 Message Queues:
 q     450 0x00004fec --rw-r--r-- andreas          usr

 IPC status from /dev/knem as of Mon Dec 19 09:33:09 1988
 T       ID       KEY           MODE          OWNER     GROUP
 Message Queues:
 q     450 0x00004fec --rw-r--r--  andreas         usr

 IPC status from /dev/knem as of Mon Dec 19 09:33:24 1988
 T       ID       KEY           MODE          OWNER     GROUP
 Message Queues:
 q     450 0x00004fec --rw-r--r--  andreas         usr

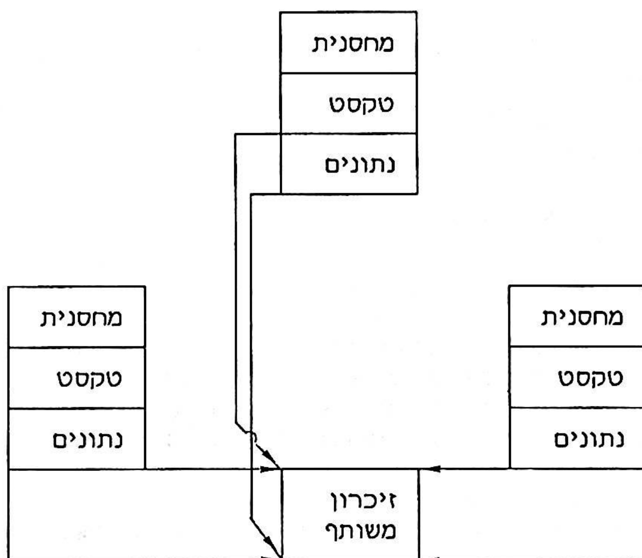
 Company name read=Corporation UNIXA
```

10.17 זיכרון משותף

זיכרון משותף (Shared Memory - SM) הינו קטע בזיכרון המרכזי של המחשב שבו יכולים להתחלק תהליכים עצמאיים. כפי שניתן לראות מתרשים 10.1, זיכרון משותף קשור לסגמנט הנתונים של התהליך והוא מופיע ככתובת בפועל, אשר שונה בכל תהליך.

הגודל המקסימלי של סגמנט משותף מוגבל רק על ידי גודל הזיכרון הפנוי. הגישה לסגמנטים המשותפים והחלפת הנתונים בין תהליכים שאינם קשורים ביניהם יכולים להעשות בצורה אמינה

בתנאי שנסנכרן את הכתיבה והקריאה לסגמנטים אלה. הסינכרון מושג בדרך-כלל בעזרת אתתים (semaphores).



תרשים 10.1 גישה לזיכרון משותף

סגמנטים משותפים בזיכרון ימשיכו להתקיים עד אשר יבוטלו באמצעות פונקצית המערכת **shmctl**, או בעזרת הפקודה **ipcrm -q** (ראה גם ipcs, ipczm, shmctl).

תכניות שצריכות לטפל בזיכרון משותף תשתמשנה במספר קובצי כותרת ובמספר קריאות לפונקציות מערכת, אשר התחביר שלהן נראה להלן:

```
#include  <sys/types.h>
#include  <sys/ipc.h>
#include  <sys/shm.h>

int shmget(key, size, flag)
key_t key;
int size, flag;
```

```
char shmat(shmid, address, flag)
int shmid;
char *address;
int flag;

int shmdt(address)
char *address;

int shmctl(shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;
```

הפונקציה **shmget** משמשת ליצירת סגמנט משותף בזיכרון, אשר גודלו ("size") מצוין בבתים והוא כולל גם את מבנה הנתונים הקשור אליו ("key"). פונקציה זו מחזירה את המספר המזהה של הזיכרון המשותף ("shmid") הקשור עם "key". ציון הדגל ("flag") יכול להיות שילוב של הערכים הבאים:

* **אופן הגישה** - הסיביות המציינות את אפשרויות הגישה לסגמנט המשותף החדש שנוצר זהות לאלו של הפונקציה **create()** (כלומר rwxrwxrwx), פרט לכך שרק ההרשאות לקריאה וכתיבה הינן בתוקף. תשע הסיביות הנמוכות של אות הסימון מגדירות את אפשרויות הגישה.

* **IPC_CREATE** - יצירה של סגמנט משותף בזיכרון אם אינו קיים עדיין; אם הסגמנט קיים, אזי איו התייחסות לאפשרות היצירה, אלא אם כן צויין הדגל **IPC_EXCL**.

* **IPC_EXCL** - אם הסגמנט המשותף קיים והשתמשו ב-**IPC_EXCL** יחד עם **IPC_CREATE**, תוחזר שגיאה.

הפונקציה **shmat** מחברת את הסגמנט המשותף המזוהה על ידי "shmid", החל מהכתובת ("address") של סגמנט הנתונים שבתהליך הקורא. אם הכתובת שווה לאפס, מערכת UNIX תחליט היכן לחבר את הסגמנט המשותף. הדגל יכול לקבל את הערכים הבאים:

0 - ערך אפס מציין שהסגמנט המשותף ניתן לקריאה ולכתיבה.
SHM_RDONLY - מציין שהסגמנט המשותף ניתן לקריאה בלבד.

לאחר סיום מוצלח, תחזיר **shmat** את הכתובת שבה חובר הסגמנט המשותף; במקרה של שגיאה, יוחזר הערך 1-.

פונקצית המערכת **shmdt** משמשת לניתוק הסגמנט המשותף ממרחב הכתובות של התהליך. היא מחזירה את הערך אפס לאחר ביצוע מוצלח ואת הערך 1- לאחר כישלון.

בעזרת הפונקציה **shmctl** ניתן לבצע בסגמנט המשותף מספר פעולות בקרה כפי שמצוין על-ידי **cmd**. הפרמטר **cmd** יכול לקבל את הערכים הבאים: **IPC_STAT**, **IPC_SET** ו-**IPC_RMID**. לדוגמה, נכתוב את הפונקציה הבאה בתכנית C:

```
shmctl(shmid,IPC_RMID,(struct shmid ds *) 0 )
```

פונקציה זו יכולה לשמש להריסת סגמנט הזיכרון המשותף יחד עם מבנה הנתונים המצורף ל-**shmid** (בסעיף הבא תוכל למצוא תכנית C המשתמשת בפונקציות המערכת המטפלות בסגמנטים).

10.18 אתתים

בגרסת UNIX System-V, אתת (semaphore) הינו מבנה נתונים שבו יכולה להתחלק קבוצה של תהליכים. כאשר מספר תהליכים מתחרים על אותו משאב (כמו קובץ או תור הודעות), ניתן לסנכרן את הפעולות בעזרת אתתים, כדי שלא יפריעו זה לזה.

אתתים מבטיחים כי רק תהליך אחד יקבל גישה בלעדית למשאב מסוים באותו זמן. הביצוע של כל התהליכים האחרים הזקוקים לאותו משאב, יושעה על ידי גרעין UNIX. לאחר שתהליך משחרר את המשאב, הוא מאפשר לתהליך מושעה להשתמש במשאב בצורה בלעדית. לעתים קרובות משתמשים באתתים בזוגות, במטרה לנעול את המשאב ולאחר מכן לשחרר אותו.

מבנה האתת ב-UNIX מכיל את המשתנים "semval", "semzcnt", "semncnt" ו-"sempid". המשתנה "semval" הינו מספר שלם לא שלילי, שערכו משתנה בהתאם להסבר על **semop** שבהמשך. "semzcnt" הינו מספר שלם קצר ללא סימן המייצג את מספר התהליכים המושעים שממתינים לכך שערכו של "semval" לגבי אתת זה יהפוך לאפס. "semncnt" הינו מספר שלם קצר ללא סימן המייצג את מספר התהליכים המושעים שממתינים לכך שערכו של "semval" יהיה גדול יותר מאשר הערך הנוכחי שלו. "sempid" מחזיק את המספר המזהה של התהליך שביצע את פעולת האיתות האחרונה באתת זה.

תכניות שצריכות להשתמש באתתים כדי לפקח על משאב מסוים, משתמשות במספר קובצי כותרת ובמספר קריאות לפונקציות מערכת. התחביר של הפונקציות הוא כדלקמן:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sem.h>

int semget(key, snumber, flag)
key_t key;
int snumber, flag;

int semop(semid, soperations, snumber)
int semid;
struct sembuf (*soperations) [];
int snumber;

int semctl(semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun{
            int val;
            struct semid_ds *buff;
            unsigned short *array;
            }arg;
```

הקריאה ל-**semget** משמשת ליצירת מערכת של אתתים בצירוף מבנה הנתונים הקשור אליו ("semid_ds"). פונקציה זו מחזירה מספר שלם חיובי הידוע כמספר המזהה של האתת ("semid"). ניתן להשתמש לאחר מכן ב-"semid" כפרמטר בקריאה לפונקצית המערכת **semop**.

הפרמטר "key" ב-**semget** מציין את שמה של מערכת האתתים, ו-"snumber" מציין את מספר האתתים במערכת. הפרמטר "flag" (משתנה דגל) יכול להיות שילוב של הערכים הבאים:

* **אופן הגישה** - הסיביות המציינות את אפשרויות הגישה למערכת האתתים החדשה שנוצרה זהות לאלו של הפונקציה **create()** (כלומר rwxrwxrwx), פרט לכך, שרק ההרשאות לקריאה וכתיבה הינן בתוקף. תשע הסיביות הנמוכות של הדגל מגדירות את אפשרויות הגישה.
* **IPC_CREATE** -יצירה של אתת בשם "key", אם הוא לא קיים עדיין; ואם האתת קיים, להתעלם מאפשרות היצירה, אלא אם צויין הדגל **IPC_EXCL**.
* **IPC_EXCL** - אם מערכת האתתים קיימת ומשתמשים ב-**IPC_EXCL** יחד עם **IPC_CREATE**, תוחזר שגיאה.

לאחר שנוצר מערך (array) של אתתים, ניתן לקרוא לפונקצית המערכת **semop** בכדי לטפל בהם. "semid" הינו המספר המזהה של האתת המוחזר על ידי **semget**, "soperations" הינו מצביע למערך המכיל את הפעולות באתת ו-"snumber" הינו מספר הרשומות במערך. המבנה של כל רשומה ב-"soperations" זהה לזה של המבנה "sembuf" שלהלן:

```
struct sembuf{
            unsigned short sem_num; /*the semaphore number*/
            short  sem_op;  /*the semaphore operation*/
            short  sem_flg; /*operation flags*/
            }
```

הקריאה ל-**semop** תפעיל את האתת שמספרו **sem_num[n]** ואשר שייך למערכת האתתים המזוהה על ידי "semid". פעולה זו מוגדרת על ידי **sem_op[-n]** ועל ידי קבוצת הדגלים **sem_flg[n]**.

תוצאות הקריאה ל-**semop** תלויות בערכים של "sem_op" ושל "sem_flg". ניתן לחלק את התוצאות לשלוש קבוצות בהתאם לערכים שהפרמטר "sem_op" יכול לקבל: מספר חיובי, מספר שלילי ואפס. אם לדוגמה "sem_op" גדול מאפס, יתוסף הערך של "sem_op" למשתנה "semval". אם התוצאה של "semflg & IPC NOWAIT" היא שקרית והערך של המשתנה "semval" קטן מאשר הערך האבסולוטי של "sem_op", יגדל הערך של המשתנה "semncnt" באחד והתהליך הקורא יושעה עד אשר הערך של "semval" יהיה גדול יותר, או שווה לערך האבסולוטי של "sem_op". כשזה יקרה, יתבצע חיסור של הערך האבסולוטי של "sem_op" מ-"semval" ואם התוצאה של "sem_flg & SEM_UNDO" תהיה אמת, יתוסף הערך האבסולוטי של "sem_op" לערך של "semadj". לפרטים על פעולות נוספות, עיין במדריך ל-UNIX בסעיף העוסק בקריאה ל-**semop**.

פונקצית המערכת **semctl** מאפשרת פיקוח על פעולת קבוצת האתתים המזוהה על ידי "semid" על פי פקודות המצוינות בפרמטר **cmd**. הפרמטר **cmd** יכול לקבל את הערכים הבאים:
GETVAL, SETVAL, GETPID, GETNCNT, GETZCNT, IPC_STAT, IPC_RMID, IPC_SET, GETALL ו-SETALL.

לפרטים נוספים על כל אחת מפקודות אלו, עיין במדריך UNIX, בסעיף העוסק ב-"semctl".

התכנית "semex.c" שלהלן, הכתובה בשפת C, תעזור לך להבין כיצד להשתמש הן בפונקציות מערכת המטפלות באתתים והן באלו המטפלות בסגמנטים משותפים בזיכרון.

התכנית יוצרת תהליך בן על ידי קריאה לפונקצית המערכת **fork**.
תהליך הבן אחראי לקריאת רשומות מהקובץ שמועבר כארגומנט לתכנית. בכל רשומה שהוא קורא, הוא משנה את שני הבתים הראשונים לאפס וכותב את המידע הזה לסגמנט המשותף.
תהליך האב קורא את הרשומה וכותב אותה לפלט הסטנדרטי.
האתתים משמשים לתיאום הפעילויות של תהליך האב ותהליך הבן בכך שהם מבטיחים שתהליך האב יקרא מהסגמנט המשותף בזיכרון רק לאחר שתהליך הבן סיים לכתוב את הרשומה.
מצורף גם תדפיס הפלט של ביצוע מעקב (trace), כדי לאפשר לך להבין את התכנית ביתר קלות.
הביצוע של התכנית "semex.c" בשליטת תכנית המעקב יפיק את הפלט הבא:

```
Just before pread of parent
Writing record number-1 away
Child about to release semaphore
Record-1=00AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Just before pread of parent
Child released semaphore
Writing record number-2 away
Child about to release semaphore
Record-2=00BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Just before pread of parent
Child released semaphore
Writing record number-3 away
Child about to release semaphore
Record-3=00CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Just before pread of parent
Child released semaphore
Calling wrbuff to stop
Time to finish program
End record found.All is OK
```

```
/* Program      : semex.c */
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
```

```
#define WRITE 0
#define STOP 1
#define ERROR (-1)
#define SMSIZE 512 *1   /* size of shared memory (SM) segment*/
#define its child (getpid() == cpid)
#define recsize 50 /* length of record to read from SM segm. */
struct sminfo {
       int wtsem;
       int rdsem;
       char *buf;
       }
char *cshm;
int ppid, cpid;
extern char *shmat();

main(argc, argv)
int argc;
char *argv[];
{
int pid, getout();
signal(SIGUSR1, getout);signal(SIGHUP, getout):
signal(SIGTERM, getout);signal(SIGINT, getout):
psminfo= &sminfo;

if( argc != 2 )
    {
    fprintf(stderr,"Usage is : progname filename\n");exit(1);
    }
ppid=getpid(); /* get this process's id ( parent id) */
allipc(); /* create two semaphores and a SM segment */

/* attach SM segment to the process address space */
if( (psminfo->buf = shmat(shmid, 0, 0) ) == (char *) ERROR )
    {
    perror("Unable to attach SM segment");exit(4);
    }

vwrite(psminfo);
    if( (pid=fork() ) == 0 )
    {
    /* child process so write data to SM segment */
    cpid=getpid();
    pwrite(psminfo); /* get buffer for writing to */
```

```
      cshm=psminfo->buf;
      sendsm(argv[1]);
      }
  else if( pid == ERROR )
      {
      perror("Unable to fork");exit(7);
      }
  else
      {
      /* its parent process so read SM segment */
      outrec();
      }
 exit(0);
 } /* end of main program */

 /* Routine : To process a record */
 sendsm(filename)
 char *filename;
 {
 int fd,nobytes,count=0;
 char rec[512];
 /* open filename to read records */
 if( (fd=open(filename, O_RDONLY) ) == ERROR )
     {
     sprintf(rec,"Error in opening filename %s",filename);
     prerr(rec);
     }
 /* loop to read three records from the file */
 nobytes=recsize;
 while( count < 3 )
     {
     if( read(fd,rec,nobytes) != nobytes )
          {
          sprintf(rec,"Error in reading file %s",filename);
          prerr(rec);
          }
     /* change first two bytes of record to zeros */
     rec[0]='0';rec[1]='0';
     /* write record to SM segment */
     wrbuff(rec,nobytes,WRITE);
     count++;
     }
fprintf(stdout,"Calling wrbuff to stop\n");
wrbuff(rec,0,STOP);
}
```

```
/* Routine : Write record to SM segment,release semaphore*/
wrbuff(buf,nobytes,cmd)
char *buf;
int nobytes, cmd;
{
char rec[100];
static int count=1;
if( cmd == WRITE )
    {
    fprintf(stdout,"Writing record number-%d away\n",count);
    memcpy(cshm, buf, nobytes); /*write record to SM segment*/
    fprintf(stdout,"Child about to release semaphore\n");
    count++;
    return(1);
    }
else if( cmd == STOP )
    {
    fprintf(stdout,"Time to finish program\n");
    memcpy(cshm,"***END***",9); /* to tell parent to stop */
    vread(psminfo);
    close(0);
    shmdt(psminfo);
    exit(0);
    }
else
    {
    sprintf(rec,"Unrecognized command in wrbuff is %d",cmd);
    prerr(rec);
    }
return(0);
}

/* Routine      : To read SM segment and output the record */
outrec()
{
char rec[10];
static int count=1;
while(1) /* infinite loop to read records */
    {
    fprintf(stderr,"\nJust before pread of parent\n");
    pread(psminfo); /* get a buffer for reading */
    strncpy(rec,psminfo->buf,9);
```

```
     if( strncmp(rec,"***END***",9) == 0 )
          {
          close(1);
          fprintf(stderr,"End record found.All is OK\n");
          cloipc();
          exit(0);
          }
     /* output contents of SM segment to standard output */
     write(1,psminfo->buf,recsize);
     vwrite(psminfo); /* release buffer for writing */
     } /* end of infinite loop */
}

/* Routine      : Print error message,tidy up and exit program */
prerr(msg)
char *msg;
{
extern int errno, sys_nerr;
extern char *sys_errlist[];

fprintf(stderr,"%s\n",msg); /* print error message */
if( errno > 0 && errno < sys_nerr )
     fprintf(stderr,"%s\n",sys_errlist[errno]);
if( its_child )
     kill(ppid, SIGUSR1); /* send signal to kill parent */
else
     kill(cpid, SIGUSR1); /* send signal to kill child */
cloipc();
exit(9);
}

/* Routine      : Cope with signals defined in "main" program */
getout()
{
if( its_child )
    {
    close(1);
    cloipc();
    }
else /* its the parent */
    {
    close(0);
```

```
    shmdt(psminfo -> buf); /*detach SM segment */
    }
exit(5);
}

/* Routine      : To create two semaphores and a SM segment */
allipc()
{
if( ( psminfo -> rdsem = semget( (key_t) ppid, 1, IPC_CREAT |
                 IPC_EXCL | 0664 ) ) == ERROR )
     prerr("Unable to create first semaphore");
if( ( psminfo -> wtsem = semget( (key_t) ppid+1, 1, IPC_CREAT |
                 IPC_EXCL | 0664 ) ) == ERROR )
     prerr("Unable to create second semaphore");

if( ( shmid = shmget( (key_t) ppid, SMSIZE, IPC_CREAT |
                 IPC_EXCL | 0664 ) ) == ERROR )
     prerr("Unable to create SM segment");
}

/* Routine      : To remove semaphore ids and structures and
                  also SM segment id and associated structures*/
cloipc()
{
semctl(psminfo->rdsem, 0, IPC_RMID, 0);
semctl(psminfo->wtsem, 0, IPC_RMID, 0);
shmctl(shmid, IPC_RMID, 0);
}

/* Routine      : To perform "semop" on a semaphore. If val is 1
increment "semval" by 1; if val is -1 suspend process until
"semval" is 1 and then decrement it by absolute value of val */

csemop(semid, val)
int semid, val;
{
struct sembuf sval;
sval.sem_num=0;
sval.sem_op=val;
sval.sem_flg=0;
if( semop(semid, &sval, 1) == ERROR)
    prerr("Unable to perform semaphore operation");
}
```

```
/* Routine     : To do P operation; lock (get) semaphore */
P(semid)
int semid;
{
csemop(semid, -1);
}

/* Routine     : To do V operation;unlock (release) semaphore*/
V(semid)
int semid;
{
csemop(semid, 1);
}

/* Routine : To get the write semaphore for the buffer */
pwrite(pshm)
struct sminfo *pshm;
{
P(pshm -> wtsem );
}

/* Routine : To get the read semaphore for the buffer */
pread(pshm)
struct sminfo *pshm;
{
P(pshm -> rdsem );
}

/* Routine : To release the read semaphore for the buffer */
vread(pshm)
struct sminfo *pshm;
{
V(pshm -> rdsem );
}

/* Routine : To release write semaphore for the buffer */
vwrite(pshm)
struct sminfo *pshm;
{
V(pshm -> wtsem );
}
```

פרק 11

כלים להפקת פרויקטים ולתחזוקתם

הקשיים בפיתוח ובתחזוקה של תוכנה ידועים מאז החלו לעסוק בנושאים אלה. מבין האמצעים הרבים שמערכת UNIX מספקת בכדי להקל על הפיתוח, התחזוקה והשחרור של פרויקטים, אפשר למנות את **make** ואת המערכת לבקרת קוד מקור (SCCS).

make הינו הכלי כדי לחולל (generate) בעזרתו את המערכת. תוכנת SCCS מאפשרת לתחזק מספר מהדורות של התוכנה בקובץ אחד. הדבר חשוב ושימושי במיוחד במערכות ייצור, אשר בהן המהדורה הנוכחית הפעילה מתוחזקת באופן שוטף, בעת אשר מפתחים מהדורות חדשות.

11.1 make

תכניות גדולות מחולקות למספר חלקים קטנים יותר כדי שניתן יהיה לנהל אותם ביתר קלות. תכניות אלו תלויות בדרך כלל בקבצי מקור שונים, אשר יכולים להיות בתת-מדריכים שונים. התלות שנוצרת בין הקבצים מקשה על המעקב והידיעה מהי הגרסה המעודכנת ביותר. הפקודה **make** מאפשרת להתגבר על קשיים אלה. מבנה ההוראה:

```
$make [-f mfile] [אופציות] [הגדרות מקרו] [קובצי מטרה]
```

הפקודה **make** מבצעת את הפקודות שב-"mfile" וכך מתבצע העדכון ב"קובצי המטרה". אם לא משתמשים באופציה **f-**, הפקודה בודקת את קובצי התיאור "makefile", "Makefile", "s.makefile", "s.Makefile", לפי סדר זה. להלן כמה מבין האופציות הנפוצות:

n- הדפס את הפקודות שיתבצעו בקובץ make, אך אל תפעיל אותן, כדי לבנות את קובצי המטרה.

q- בתשובה לשאלה לקובץ make, ניתן לדעת אם קובץ המטרה מעודכן או לא. ערך חוזר השווה לאפס משמעותו שהקובץ מעודכן וערך השונה מאפס משמעותו שהקובץ אינו מעודכן.

t- עדכן את זמני השינוי בקובצי המטרה.

r- אל תשתמש בכללים המובנים הנמצאים בטבלאות הפנימיות.

-d הדפס מידע מפורט על הקבצים כדי לעזור בתהליך הניפוי.

-f mfile השתמש ב-"mfile" כקובץ התיאור (במקום ברירת המחדל "makefile", "Makefile", "s.makefile", "s.Makefile")

11.2 כתיבת קובצי make

המשתמש צריך ליצור את קובץ **make** לפני שהוא משתמש בפקודה **make**. קובץ זה מכיל מידע על יחסי תלות בין קבצים, הגדרות מקרו ופקודות לביצוע. השורות המכילות את המידע על יחסי התלות מציינות, למעשה, כיצד יש ליצור את קובץ המטרה ממספר קבצים. אם תאריך העדכון של קבצים אלה מאוחר יותר מאשר זה של קובץ המטרה, קובץ המטרה יופק מחדש בהתאם לפקודות שתהיינה לאחר השורה המכילה את המידע על יחסי התלות. המבנה הפשוט של קובץ make נראה כך:

```
target : dependency-list   רשימת תלות : קובץ מטרה
TAB-character executable command   TAB-פקודה לביצוע תו
```

להלן קובץ **make** (makefile) פשוט:

```
#
# makefile example-1
#
math.x : add.o mult.o sub.o div.o
        cc -o math.x add.o mult.o sub.o div.o -lm

func.x : add.o sub.o
        cc -o func.x add.o sub.o -lm -lcurses

add.o  : add.c
        cc -c add.c

mult.o : mult.c
        cc -c mult.c

sub.o  : sub.c
        cc -c sub.c

div.o  : div.c
        cc -c div.c
```

"math.x" ו-"func.x" הינם קובצי המטרה. "add.o", "mult.o", "sub.o" ו-"div.o" יוצרים את רשימת התלות של "math.x" ו-cc הינה הפקודה לביצוע; יש להקדים את **cc** בטאבולטור. קובץ **make** הנראה לעיל נשמר בשם "makefile". פקודת **make** תפיק את קובץ המטרה "math.x" כפי שנראה להלן:

```
$make
cc -c add.c
cc -c mult.c
cc -c sub.c
cc -c div.c
cc -o math.x add.o mult.o sub.o div.o -lm
```

שים לב שקובץ המטרה "func.x" לא הופק. הסיבה לכך היא, שכאשר מבצעים אתת הפקודה **make** ללא שמות קובצי מטרה, היא תיצור רק את קובץ המטרה הראשון המוזכר בקובץ **make**. לאחר ביטול קובצי היעד "add.o", "mult.o", "sub.o" ו-"div.o" ניתן להפיק את קובצי המטרה "math.x" ו-"func.x" בעזרת פקודת **make**, כפי שנראה להלן:

```
$make math.x func.x
cc -c add.c
cc -c mult.c
cc -c sub.c
cc -c div.c
cc -o math.x add.o mult.o sub.o div.o -lm
cc -o func.x add.o sub.o -lm -lcurses
```

בדרך זו הפקנו את שני קובצי המטרה. בשלב זה נקיש את הפקודה

```
$make math.x func.x
```

תכנית השירות **make** תדפיס את הפלט הבא:

```
'math.x' is up to date
'func.x' is up to date
```

הסיבה לכך היא, שאף קובץ מאלה הנמצאים ברשימת התלות לא עבר שינוי. אם נקיש **make** לאחר שהקובץ "mult.c" עבר שינוי באמצעות תכנית עריכה, או באמצעות הפקודה "touch mult.c", נקבל את הפלט הבא:

```
$make
cc -c mult.c
cc -o math.x add.o mult.o sub.o div.o -lm
```

שים לב שזהו מספר הפקודות המינימלי הדרוש להפקת קובץ המטרה "math.x".

11.3 ברירות המחדל של סיומות קובצי make

מהדיון הקודם צריך להיות ברור שכדי להפיק קובץ מטרה, משתמשת **make** בקובץ תיאור שהמשתמש מספק (makefile), בשמות קבצים ובתאריכי השינוי האחרונים שלהם כפי שרשום במערכת הקבצים. אולם, תכנית השירות make משתמשת גם במספר כללים מובנים. המידע כיצד לשנות קובץ בעל סיומת אחת לקובץ בעל סיומת אחרת מאוחסן בטבלה פנימית. רשימת הסיומות המוגדרות כברירות המחדל ניתנת בטבלה 11.1.

טבלה 11.1 טבלת ברירת המחדל של הסיומות

סוג הקובץ	סיומת
קובץ יעד (object file)	.o
קובץ מקור בשפת C	.c
קובץ מקור בפורטרן	.f
קובץ מקור ברטפור (Ratfor)	.r
קובץ מקור באפי (Efi)	.e
קובץ מקור באסמבלר	.s
דקדוק המקור Lex	.l
דקדוק המקור Yacc-C	.y
דקדוק המקור Yacc-Ratfor	.yr
דקדוק המקור Yacc-Efi	.ye

כך, על ידי ניצול המידע הפנימי האגור ב-make, ניתן לכתוב מחדש את הדוגמה "makefile example-1" כפי שנראה להלן:

```
#
# makefile example-2
#
```

```
math.x : add.o mult.o sub.o div.o
        cc -o math.x add.o mult.o sub.o div.o -lm

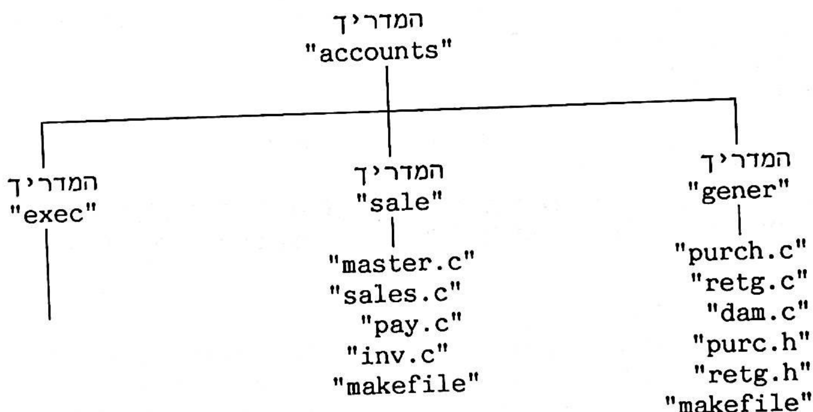
func.x : add.o sub.o
        cc -o func.x add.o sub.o -lm -lcurses
```

11.4 פקודת מקרו בקובצי make

בתכנית השירות **make** ישנו אמצעי לביצוע פקודות מקרו. ניתן להגדיר פקודות מקרו בקובץ **make** וניתן להעביר אותן כארגומנטים לקובץ **make**. בקובץ **make** מוסיפים לאחר שם המקרו את הסימן = (שווה) ורצף של תווים. ניתן להפעיל את המקרו בעזרת סימן הדולר. שם מקרו אשר ארוך יותר מתו אחד חייב להכתב בסוגריים. פקודות המקרו המועברות כארגומנטים יבטלו את המשמעות של פקודות מקרו עם שמות זהים הנמצאים בתוך הקובץ.

כאשר מפתחים מערכת מורכבת, כדאי שלכל יישום יהיה מדריך מקור משלו. לכל מדריך יהיה קובץ **make** משלו וכל קובץ כזה יכיל את כל המגוון של הפונקציות המשמשות לתחזוקה.

הדוגמה הבאה מציגה את אשר הוסבר לעיל. קבצים בחבילת תוכנה להנהלת חשבונות יכולים להיות מפוזרים במספר מדריכים כפי שנראה בתרשים 11.1.



תרשים 11.1 קובצי חבילת תוכנה להנהלת חשבונות

קבצים בעלי הסיומת ".c" ו-".h" מייצגים בהתאמה קובצי מקור בשפת C וקובצי כותרת. שני קובצי make נבנו להפקת המערכת הנ"ל: קובץ "makefile example 3" הנמצא במדריך "sale" וקובץ "makefile example 4" הנמצא במדריך "gener". להלן התוכן של שני קובצי make אלה:

```
#
# makefile example-3 in the sale directory
#

TFLAGS = -lm
DIR = ../gener
PURCH = $DIR/purch.o $DIR/retg.o $DIR/dam.o

intacc  : master.o sale.o pay.o inv.o
          ( cd ../gener;make )
          cc -o intacc master.o sale.o inv.o pay.o $(P
          $(TFLAGS)

install : intacc
          cp intacc ../exec
          rm intacc

print   : lp master.c

#
# makefile example-4 in gener directory
#

gener   : purch.o dam.o retg.o
purch.o : purch.h
retg.o  : retg.h
```

"TFLAGS", "DIR" ו-"PURCH" הינם דוגמאות של פקודות מקרו. ביצוע של פקודת **make** במדריך "sale" יתן את התוצאות הבאות:

```
$make
cc -o -c master.c
cc -o -c sale.c
cc -o -c pay.c
cc -o -c inv.c
```

```
(cd ../gener;make)
cc -o -c purch.c
cc -o -c dam.c
cc -o -c retg.c
cc -o intacc master.o sale.o inv.o pay.o
      ../gener/purch.o ../gener/retg.o
      ../gener/dam.o -lm
```

לאחר פקודת **make** הקודמת תבואנה ברצף הפקודות הבאות:

```
$touch master.c
$make "TFLAGS = -lm -lcurses"
```

הפלט של פקודת **make** יהיה עכשיו:

```
cc -o -c master.c
( cd ../gener;make )
cc -o intacc master.o sale.o inv.o pay.o
      ../gener/purch.o ../gener/reg.o
      ../gener/dam.o -lm -lcurses
```

שים לב שמתבצע הידור של הקובץ "master.c" מכיון שהוא הקובץ היחיד שעבר שינוי באמצעות הפקודה **touch**. כאן אנו רואים שוב שזהו מספר הפקודות המינימלי הדרוש להפקת גרסה מעודכנת של הקובץ לביצוע "intacc". נקודה נוספת שראוי לשים לב אליה היא, שהערך של המקרו "TFLAGS" הוחלף בארגומנט "-lm -lcurses".

קובץ המטרה "install" הוגדר ב-"makefile example-3". לפיכך, אם נכתוב את ההוראה

```
$make install
```

היא תדפיס את הפלט הבא:

```
cp intacc ../exec
rm intacc
```

הפקודה מעתיקה את הקובץ "intacc" למדריך "exec" ומבטלת אותו במדריך "sale". "print" הוגדרה באופן דומה בקובץ "makefile example-3" ולכן הפקודה **make print** תדפיס את קובץ המקור "master.c" הנמצא במדריך "sale". בצורה זו אפשר להכניס פונקציות תחזוקה רבות לקובץ make.

11.5 הכללים המובנים של make

עבור כל סיומת הנמצאת בטבלה 11.1 קיים ב-**make** כלל פנימי המגדיר את הפקודות שיבוצעו כדי לשנות קובץ עם סיומת אחת (לדוגמה, קובץ ".c") לקובץ עם סיומת אחרת (לדוגמה, קובץ ".o").

ההוראה **make** משתמשת גם במספר פקודות מקרו מוגדרות מראש. המקרו "CC", לדוגמה, מציין את שם המהדר שהינו ברירת המחדל, והמקרו "CFLAGS" מציין את הדגלים שהינם ברירת המחדל למהדר "CC". המשתמש יכול להגדיר מחדש את פקודות המקרו האלו.

קיימות פקודות מקרו נוספות המוגדרות מראש:

- **$*** שם קובץ המטרה ללא הסיומת.
- **$@** השם המלא של קובץ המטרה.
- **$<** שם הקובץ שמתחיל את הפעולה.
- **$?** רשימת הנתמכים שאינם יותר בתוקף.

לפני שהיא מעבדת את הקבצים לפי כללים מסוימים, בודקת **make** איזה כללים לשינוי סיומות נמצאים במסגרת הכלל ".SUFFIXES".

בדוגמה שלהלן השתמשנו במספר רעיונות שהועלו כאן. בקובץ make זה מתבצע הידור של מספר תכניות הכתובות ב-SVS-Fortran. תכניות אלו מקושרות לאחר מכן עם מודולים בשפת C כדי להפיק קובץ ביצוע הנקרא "master". שים לב במיוחד להגדרה של כללים חדשים (לדוגמה המרת ".c" ל-".obj") ולכלל ".SUFFIXES". בחן בקפדנות את נתוני המעקב (trace) המופקים על ידי הפקודה "make -n".

```
## makefile example-5
#


.SUFFIXES .w .o .c .for .obj
# the target file is master and ex105 is a fortran program
# ex124 is a C program
master: ex105.obj ex124.obj
        ulinker -l out.o ex105.obj /usr/lib/ftnlib.obj /usr/lib/
paslib.obj


        cc out.o ex124.obj -o master
# new rule which converts a ".c" file to a ".obj" instead of ".o"
.c .obj:
```

```
        cc -c $<
        cp $*.o $*.obj
# new rule which converts a ".for" file to a ".obj" file
.for .obj:

cp $< temp.c
cc -P temp.c
mv temp.i $*.k.for
fortran $*.k.for
code *.k.i
mv $*.k.obj $*.obj
rm -f temp.c
```

הפקודה

$make -n

תפיק את נתוני המעקב הבאים:

```
cp ex105.for temp.c
cc -P temp.c
mv temp.i ex105.k.for
fortran ex105.k.for
code ex105.k.i
mv ex105.k.obj ex105.obj
rm -f temp.c
cc -c ex124.c
cp ex124.o ex124.obj
ulinker -l out.o ex105.obj /usr/lib/ftnlib.obj /usr/lib/paslib.obj
cc out.o ex124.obj -o master
```

11.6 מערכת לבקרת קוד מקור (SCCS)

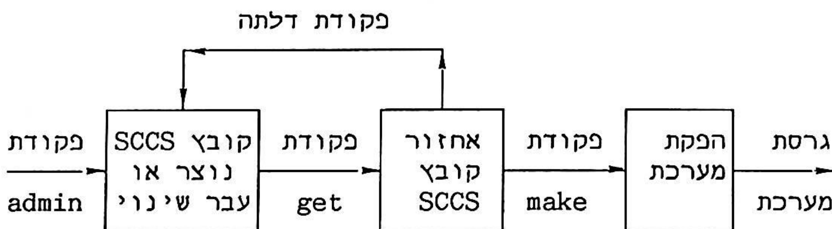
במהלך מחזור החיים של תכניות הן עוברות שיפורים, ניפוי שגיאות והוצאה של מספר גרסאות. התכנית לבקרת קוד מקור (להלן Source Code Control System - SCCS) הינה מערכת של כלים שניתן לעקוב בעזרתה אחר שלבי ביניים רבים של התכנית.

התוכנה SCCS מאפשרת למעשה למשתמש לתחזק מספר גירסאות שונות של התכנית בקובץ פיסי אחד בדיסק. קובץ זה מכיל מידע המספיק ליצירה מחדש, במגבלות מסוימות, של גרסאות ישנות. על פי בקשה ניתן לקבל גרסה מסוימת ומכיון שרק ההבדלים בין גרסה זו לאחרת נשמרים, הדבר חסכוני מבחינת המקום בדיסק.

מנהל הפרויקט יכול לציין, בעזרת סדרה של פקודות, אילו פעולות מותרות בקובץ SCCS מסוים.

11.7 העברת תכניות לפיקוח של SCCS

הפקודה **admin** יכולה לשמש להעברת קובץ מקור מסוים לפיקוח של SCCS. הגרסה הראשונה של קובץ SCCS היא בדרך כלל 1.1, אשר משמעותו "גרסה 1 רמה 1". כל שינוי שיעבור קובץ SCCS מכאן ואילך ידוע בשם דלתה (delta) וכך, כל פקודת דלתה מעלה את מספר הגרסה, או את מספר הרמה, או את שניהם כאחד.



תרשים 11.2 מחזור SCCS טיפוסי

כדי להעביר את קובץ המקור "time.c" לפיקוח של SCCS, נשתמש בפקודה הבאה:

```
$admin -itime.c s.time.c
no id keywords (cm7)
```

הפקודה **admin** מדפיסה את ההודעה "no id keywords (cm7)" ומסיימת. אפשר לבטל את קובץ המקור ("time.c" בדוגמה שלעיל) מכיון שתוכנו נמצא עכשיו בפיקוח SCCS. שים לב, בשלב זה כל קובץ הנמצא בפיקוח של SCCS חייב להכיל את הקידומת ".s". כמו כן, אין להשאיר רווח בין האופציה **-i** לשם הקובץ.

מהרגע שבו הקובץ נמצא בפיקוח של SCCS, ניתן להשתמש בפקודה **get** בכדי לאחזר כל גרסה שלו למטרת קריאה בלבד. לדוגמה:

```
$get s.time.c
1.1
31 lines
No id keywords (cm7)
```

משמעות ההודעות הן שהקובץ "time.c", שהופק מ-"s.time.c", הינו: "גרסה 1 רמה 1" ואורכו 31 שורות.

כדי לשנות את קובץ המקור, השתמש בפקודה הבאה, אשר מפיקה קובץ לקריאה וכתיבה מ-"s.time.c".

```
$ get -e s.time.c
1.1
new delta 1.2
31 lines
```

הפקודה גם מפיקה, כמוצר לוואי, את הקובץ "p.time.c", אשר מציין "מי עושה מה" לקובץ "s.time.c". תוכל עכשיו להשתמש בתכנית עריכה של UNIX (כמו **vi**) כדי לשנות את הקובץ.

```
$vi time.c
```

כדי לשלב את השינויים כדלתה חדשה לתוך הקובץ "s.time.c" (ליצור למשל, גרסה חדשה של "time.c") השתמש בפקודה הבאה:

```
$delta s.time.c
comments?
1.2
3 inserted
0 deleted
31 unchanged
```

נתוני המעקב שבדוגמה לעיל מראים את השינויים שנעשו. המשתמש חייב למלא את שדה ההערות, כדי לציין אילו שינויים נעשו.

ניתן לבצע סיעוף משנה ברמה מסוימת של הגרסה. לדוגמה, סיעוף בגרסה 1.2 יכול להיות 1.2.1.1, אשר ניתן לפרש אותו כך: גרסה 1, רמה 2, סעיף 1 ומספר 1. כדי ליצור סעיף, השתמש תחילה בפקודה הבאה כדי לאחזר את הגרסה האחרונה של "time.c":

```
$get -b -e s.time.c
1.2
new delta 1.3
34 lines
```

שים לב שניתן להשתמש באופציה **-b** בפקודת **get** רק אם הפקודה **admin**, אשר העבירה את "time.c" לשליטת SCCS, הכילה את

האופציה -fb (כלומר: "admin -fb -itime.c s.time.c"). בצע את השינויים הדרושים בתכנית והשתמש בפקודה הבאה כדי ליצור סעיף חדש:

```
$delta -rl1.2.1.1 s.time.c
```

כאשר יש צורך בשינויים מהותיים ויש ליצור גרסה חדשה, ניתן להעלות את מספר הגרסה ב-1 בעזרת הפקודה הבאה:

```
$get -e -r2 s.time.c
1.3
new delta 2.1
35 lines
```

לאחר ביצוע השינויים הדרושים ב-"time.c" באמצעות תכנית עריכה, תיצור הפקודה הבאה את הגרסה החדשה 2.1:

```
$delta s.time.c
comments?
No id keyword (cm7)
2.1
18 inserted
0 deleted
35 unchanged
```

11.8 סמלים מיוחדים ב-SCCS

"No id keyword (cm7)" היא הודעת אזהרה. ניתן לקבל מידע מפורט על הודעות האזהרה והשגיאה בעזרת הפקודה **help**. לדוגמה, כדי לקבל מידע הנוגע לאזהרה "cm7", השתמש בפקודה הבאה:

```
$help cm7
```

קוד ההודעה "cm7" מזהיר את המשתמש ש-SCCS אינו יכול לסמן גרסה מסוימת. כדי למנוע את הודעת האזהרה ניתן להכניס לקובץ המבוקר על ידי SCCS מספר סמלים מיוחדים:

%I% מספר הגרסה.
%M% שם המודול.
%Z% המחרוזת (#)@.
%E% תאריך השינוי.
%D% תאריך האחזור.
%W% שווה-ערך ל-%I%, %M% ו-%Z%.

בחירה בסמלים אלה נעשית בדרך כלל בתחילת תכנית המקור של המשתמש בשפת C, באמצעות ההכרזה הבאה:

```
static char SCCSid[] = "%W% - %E% - A. Jones";
```

לאחר שנשתמש בפקודה **get** לקובץ SCCS, אפשר להחליף את ההכרזה הבאה על ידי המחרוזת הבאה:

```
static char SCCSid[] = "@(#)time.c 1.1 - 88/11/23 A. Jones";
```

במחרוזת זו "time.c" הינו שם התכנית, "1.1" הינו מספר הגרסה של התכנית ו-"88/11/21" מציין את תאריך השינוי. הכרזה כזו לא רק תבטל את הודעת השגיאה "cm7", אלא תספק גם מידע חשוב לפקודה **what**. פקודה זו סורקת את הקובץ כדי למצוא את המחרוזת "@(#)". אם תופעל הפקודה **what** על התכנית "time.c", יוצג המידע הבא:

```
$what time.c
time.c: time.c  1.1 - 88/11/23 A. Jones
```

make-ו SCCS 11.9

כפי שהוזכר בתחילת הפרק, הפקודה **make** תחפש גם את הקבצים "s.makefile" או "s.Makefile". קבצים אלה הם קובצי תיאור של SCCS ש-**make** יכולה לזהות אותם ולטפל בהם לפי כללים מסוימים.

הדוגמה הפשוטה שלהלן מבהירה כיצד ניתן לגרום לכך ש-SCCS ו-**make** יפעלו יחדיו. בתחילה נעביר את התכנית לשליטת SCCS. בהנחה שהתכנית נקראת "swap.c", נשתמש בפקודה **admin** הבאה כדי להעביר אותה לשליטת SCCS:

```
$admin -iswap.c s.swap.c
$rm swap.c
```

עכשיו נכתוב קובץ **make** כמו זה שלהלן:

```
sccsid="%W% - %E% - A.Jones"
swap      : swap.o
            cc -o swap swap.o
```

נעביר את קובץ ה-**make** לשליטת SCCS:

```
$admin -imakefile s.makefile
```

הפקודה **make** תיצור עכשיו קובץ לביצוע הנקרא "swap" ותפיק את נתוני המעקב הבאים:

```
$make
+ get s.makefile
1.1
3 lines
      get -p s.swap.c > swap.c
1.1
129 lines
      cc -O -c swap.c
      rm -f swap.c
      cc -o swap swap.o
```

שים לב, שפקודת **make** מזהה, באמצעות כללים מסוימים, שקובץ "swap.c" הוא בשליטת SCCS. בשלב הבא היא משתמשת בפקודה **get** עם האופציה **p-**, כדי להפיק את "swap.c" מתוך "s.swap.c". לאחר מכן היא יוצרת קובץ לביצוע הנקרא "swap" (הפקודה **get** עם האופציה **p-** מאחזרת טקסט מתוך קובץ SCCS ומעבירה אותו לפלט הסטנדרטי).

11.10 פקודות SCCS נוספות

לפקודה **admin** יש אופציות שימושיות רבות, אשר כדאי לבחון אותן מקרוב. לדוגמה, הפקודה שלהלן תוסיף את המשתמש "paul" לרשימת המשתמשים שרשאים לבצע שינויים (פקודות דלתה) בקובץ "s.time.c" ותוציא את המשתמש "peter" מרשימה זו:

```
$admin -apaul -epeter s.time.c
```

הפקודה הבאה תאפשר לבצע מספר שינויים בו-זמנית לקובץ:

```
$admin -fj s.time.c
```

הפקודה הבאה תנעל את גרסה 3 בצורה כזו, שלא ניתן יהיה לבצע בה פקודות דלתה נוספות, או אחזור לצורך עדכון.

```
$admin -fl3  s.time.c
```

ניסיון לאחזר גרסה נעולה יגרום להודעת השגיאה הבאה:

```
$get -e s.time.c
3,2
ERROR [s.time.c] release '3' locked against editing (c023)
```

כדי לקבל את רשימת הגרסאות הנעולות של התכנית "time.c", אפשר להשתמש בפקודה הבאה:

```
$prs -d:LK: s.time.c
3
```

הפלט:

הפקודה **prs** לעיל מציינת שרק גרסה **3** ננעלה. כדי לשחרר את גרסה **3** נשתמש בפקודה הבאה:

```
$admin -dl3 s.time.c
```

לבסוף, יש פקודות שימושיות נוספות בעת עבודה ב-SCCS. להלן רשימת הפקודות:

prs מדפיסה מידע הקשור לקובץ SCCS כמו פקודות הדלתה של הקובץ:

```
$prs s.time.c
```

sact מדפיסה את הפעילות הנוכחית הקשורה לקובץ SCCS:

```
$sact s.time.c
no outstanding deltas for : s.time.c
```

sccsdiff משווה ומדפיס את ההבדלים בין שתי גרסאות בקובץ SCCS:

```
$sccsdiff -r1.1 -r1.2 s.time.c
```

cdc משנה את תיאורי הדלתות הקשורות לקובץ SCCS (בקשות לשינוי והערות):

```
$cdc s.time.c
```

rmdel מבטלת דלתה בקובץ SCCS. הדלתה שעומדת להיות מבוטלת חייבת להיות האחרונה בענף שלה.

```
$rmdel -r1.1.1.1 s.time.c
```

unget מבטלת את האחזור הקודם של קובץ SCCS:

```
$get -e s.time.c
1.3
new delta 1.4
$unget -r1.4 s.time.e
```

פרק 12

תקשורת מקומית - שירותי דואר אלקטרוני

מערכת UNIX כוללת אמצעים סטנדרטיים אחדים המאפשרים למשתמשי המערכת להתקשר ביניהם ובינם לבין מנהלן המערכת בדרך של שירותי דואר אלקטרוני. בפרק זה נתאר אמצעים אלה ונעסוק בפקודות המאפשרות תקשורת מקומית בלבד, באותה מערכת מחשב. בפרק 13 תמצא מאפיינים של תקשורת בין מערכות UNIX שונות.

12.1 העברת הודעות למשתמשים

מערכת UNIX כוללת מרכיבים בסיסיים של מערכת דואר אלקטרוני. בעזרת פקודת **mail** ניתן לשלוח הודעות למשתמשי UNIX. הפקודה כוללת את שם, או שמות הנמנעים, כפי שהם מוכרים במערכת.

```
$mail user-name(s)
```

לדוגמה, כדי לשלוח הודעות למשתמשים "anne" ו-"andreas" צריך להקיש את הפקודה הבאה:

```
$mail anne andreas
```

בשורה שלאחרי פקודה זו הקש את תוכן ההודעה. גוף ההודעה צריך להיות שורה אחת לפחות. סיים את הודעתך בתו end-of-file (כלומר הקשת CONTROL-D).

גם אם נשלחת הודעה למשתמש שאינו מוכר ל-UNIX (לדוגמה, כאשר אינך מקיש נכון את שם המשתמש), אין היא הולכת לאיבוד אלא נשמרת בקובץ הנקרא "dead.letter". ניתן לאחזר את ההודעה ולשדר אותה מחדש בדרך זו:

```
$mail user-name < dead.letter
```

12.2 קריאת הודעות

הודעות הנשלחות למשתמש אחר מאוחסנות בדרך כלל במדריך ההודעות של המשתמש. מדריך ההודעות נקבע לפי משתנה הסביבה "MAIL". משתנה זה נמצא בדרך כלל בקובץ "$HOME/.profile" ומוגדר כך:

```
MAIL=/usr/mail/user-login-name
```

כאשר המשתמש נכנס למערכת הפקודה **mail** מודיעה לו אם ממתינות עבורו הודעות. במקרה שיש עבורו הודעות, הוא מקבל את ההודעה הבאה:

```
you have mail
```

כדי לקרוא את ההודעות נשתמש בפקודה **mail** במבנה הבא:

```
$mail [-options(s)] [-f file]
```

אופציות:

-p	להדפיס את תכולת תיבת הדואר, מבלי לבקש מהמשתמש להתייחס להודעה זו או אחרת.
-r	לשנות את הסדר שבו ההודעות מודפסות, מאחרון-נכנס-ראשון-יוצא (LIFO) לראשון-נכנס-ראשון-יוצא (FIFO).
-f file	להשתמש בשם קובץ מסוים להודעות ולא בברירת המחדל, שנקבעת על ידי המשתנה MAIL.

לאחר שהוצגה הודעה על המסך מקובץ הדואר של הנמען המשתמש מתבקש להתייחס אליה על ידי הצגת הסימן "?". להלן כמה מבין התשובות הנפוצות על ידי המשתמש:

<CR>	הדפס את ההודעה הבאה. (<CR> הינו קיצור של Carriage Return).
d<CR>	בטל את ההודעה הנוכחית והצג את ההודעה הבאה.
q	החזר את ההודעה לקובץ הדואר וסיים.

12.3 תקשורת בהידברות בין משתמשים

בנוסף למשלוח דואר אלקטרוני ולקבלתו באמצעות הפקודה **mail**, יכולים המשתמשים להשתמש בפקודה **write**, כדי לקיים תקשורת בנוסח של דו-שיח, או הדברות (interactive communication):

```
$write user-name [tty-name]
```

לאחר הפקודה **write** ושם הנמען יש להקיש את תוכן ההודעה ששולחים ולסיים בהקשת CONTROL-D. האופציה **tty-name** מציינת את שם המסוף ויש להשתמש בה כאשר שני אנשים או יותר נכנסים למערכת בשם זהה ממסופים שונים.

12.4 קבלת הודעות במסוף שלך

במקרים מסוימים (כמו למשל בזמן עבודה או הדגמה) אינך רוצה שיפריעו לך. לשם כך עליך לחסום את המסוף ולמנוע ממשתמשים אחרים לשלוח אליך הודעות. גם תוכל לשחרר את המסוף ולהתיר קבלת הודעות. ניתן לעשות זאת בפקודה **mesg** בעלת המבנה הבא:

```
$mesg [n or y]
```

לדוגמה, כדי למנוע ממשתמשים אחרים לשלוח הודעות למסוף שלך הקש

```
$mesg n
```

כדי לחדש את קבלת ההודעות הקש

```
$mesg y
```

12.5 משלוח הודעה לכל המשתמשים

ניתן להשתמש בפקודה **wall** כדי לשלוח הודעה לכל המשתמשים הפעילים כרגע במערכת. מסיבה זו, במערכות UNIX רבות רק מנהלן המערכת יכול לבצע את הפקודה. לעתים קרובות המנהלן משתמש בפקודה (ראה גם את הדוגמה להלן) כדי לשלוח הודעת אזהרה לכל המשתמשים לפני שהוא מכבה את המערכת, או מפסיק את פעולתה:

```
$echo "Please logout.  System going down
        in 3 minutes" | /etc/wall
```

אפשרות אחרת היא להכניס תחילה את ההודעה לקובץ (לדוגמה temp) ואז להשתמש בפקודה **wall** כדי לשלוח אותה.

```
$/etc/wall < temp
```

12.6 שירות תזכורות פרטי

ניתן להפעיל שירות תזכורות יומי באמצעות הפקודה **calendar**. תחילה עליך ליצור קובץ "calendar" ולמקם אותו במדריך הבית. קובץ זה מכיל שורות הדומות לאלו הנראות בדוגמה שלהלן:

```
Thu Feb 15    - Database demonstration at 13.00.
Feb 16        - Get present for birthday party.
19/23         - Development of screen generator.
```

לאחר מכן הצב את הפקודה **calendar** בקובץ "$HOME/.profile". בכל פעם שתכנס למערכת, תתבצע הפקודה **calendar** באופן אוטומטי ותציג שורות מתוך קובץ "calendar" המכילות את התאריך של אותו יום, או של היום שלאחריו.

במערכות UNIX ישנן צורות שונות לפקודה **calendar**. חלקן מדפיסות הודעות בפלט הסטנדרטי (הצג של המשתמש). אחרות מנתבות את ההודעה לתיבת הדואר של המשתמש שאת תכולתה ניתן לקרוא בעזרת הפקודה **mail**. כאשר משתמשים בפקודה **calendar** יחד עם האופציה מינוס (- **$calender**), ייבדקו קובצי **calendar** של כל משתמשי המערכת ויישלחו ההודעות המתאימות לתאי הדואר האלקטרוניים שלהם. פקודה זו מופעלת בדרך כלל על ידי **cron** (ראה סעיף **12.9**).

12.7 מידע על אירועים

ניתן לדווח למשתמשים על חדשות הנוגעות למערכת המחשב או לארגון באמצעות הפקודה **news**. את ההודעות שעומדות להשלח למשתמשים יש לאחסן במדריך "/usr/news". כל קובץ במדריך זה הינו פריט חדשות נפרד, אשר תוכנו קשור לאותו פריט. הפקודה **news** (ראה גם את הדוגמה להלן) נמצאת בדרך כלל בקובץ "/etc/profile" כדי שפריטי החדשות ייבדקו מייד לאחר שהמשתמש עבר את תהליך הכניסה למערכת.

```
if [ -r  /usr/bin/news ]
 then
    news -n
fi
```

הפקודה **news** היא בעלת המבנה הבא:

```
$news [options] [news-item(s)]
```

אופציות:

- **n-** הדפס רק את שמות פריטי החדשות הנוכחיים. המערכת יודעת אילו פריטים הינם "טריים" (לא הוצגו עדיין), מכיון שזמן הצגת החדשות נמצא בקובץ ".newstime" שבמדריך הבית ("HOME").
- **a-** הצג את כל פריטי החדשות (כולל אלה שהוצגו כבר).
- **s-** הדפס את מספר פריטי החדשות הנוכחיים שקיימים במדריך.

12.8 ביצוע פקודות בזמנים מוגדרים

הפקודה **at** משמשת לביצוע פקודות בזמן מוגדר:

```
$at [time] [date] [filename]
```

האופציה **time** יכולה להכיל מאחת עד ארבע ספרות. עם נכתוב עד שתי ספרות - הן תייצגנה שעות בלבד. אם נכתוב שלוש או ארבע ספרות הן תייצגנה שעות ודקות. ישנה גם אופציה להוסיף לאחר הספרות את התו **m** כדי לציין חצות, **n** - ערב, **a** - a.m. ו-**p** - p.m.

האופציה **date** יכולה לציין את היום בשבוע, או את שם החודש שלאחריו מספר היום. בדוגמאות שלהלן, הסימן **$** מציין, כרגיל, את מנחה ה-UNIX ‏(prompt).

דוגמאות להוראות ביצוע בזמנים מוגדרים:

* ב-4 אחה"צ (4 p.m.) ב-27 ליולי, בצע את קובץ הפקודות (shell script) "master.sh":

```
$at 1600 Jul 27 master.sh
```

אפשר גם לכתוב:

```
$at 4p Jul 27 master.sh
```

* ב-10 אחה"צ (10 p.m.) ביום שלישי בצע את קובץ הפקודות "backup.sh":

```
$at 10p Thursday backup.sh
```

* ב-0924 בצע את הפקודה **who** ושלח את הפלט שלה לתא הדואר של המשתמש "andreas":

```
$at 0924
who ¦ mail andreas
CONTROL-D
```

בכל הדוגמאות הללו, יוצרת הפקודה **at** עותק של הקובץ ושל משתני הסביבה הרלוונטיים "/usr/spool/cron/atjobs". לגבי הדוגמה השלישית למעלה, הקובץ שיווצר במדריך "atjobs" יהיה הקובץ הבא:

```
at job
export HOME; HOME='/usr/acct/andreas'
export LOGNAME; LOGNAME='andreas'
export MAIL; MAIL='/usr/mail/andreas'
```

```
export PATH; PATH=':/bin:/bin:/usr/bin:/usr/ucb:/etc'
export PS1; PS1='next > '
export SHELL; SHELL='/bin/sh'
export TERM; TERM='wy85c'
#       @(#).prot          1.1
cd /usr/lib
ulimit 524288000
umask 0
who | mail andreas
```

12.9 ביצוע פקודות על פי לוח זמנים

כאשר המערכת עוברת להיות מערכת רבת-משתמשים, נוהגים להפעיל את "שד השעון" **cron**. השד **cron** מתעורר בצורה מחזורית (כל דקה) וקורא פקודות מתוך קובץ "/usr/lib/crontab". בקובץ זה מציינים את הזמנים בהם יש לבצע פקודות מסוימות. להלן, כדוגמה, מספר שורות מתוך הקובץ "crontab" שנמצא במערכת:

```
0 1 * * * /usr/bin/calendar -
35 0 * * 1 /etc/errstop; /bin/cp /usr/adm/errfile /usr/adm/ oerrfile;
/bin/cp /dev/null /usr/adm/errfile
35 1 * * 1 /etc/cleanup > /dev/null 2>&1
09,19,29,39,49,59 * * * * /usr/bin/Apcron > /dev/null
30 18 * * 1,2,3,4,5 tar c /usr/acct
00 08 * * * rm -f /usr/spool/uucp/LCK*
0,5,10,20,25,30,35,40,45,50,55 * * * * /usr/lib/uucp/uudemon.hr> /dev/null
0 4 * * * /usr/lib/uucp/uudemon.day > /dev/null
30 5 * * 1 /usr/lib/uucp/uudemon.wk > /dev/null
```

כל שורה בקובץ "crontab" מורכבת משישה שדות. בטבלה 12.1 ניתן לראות את מבנה השורה ואת הערכים המותרים בכל שדה.

טבלה 12.1 שדות בשורה של קובץ "crontab"

Field number	1	2	3	4	5	6
Represents **מייצג**	Minutes	Hour	Day of month	Month of year	Day of week	Actual command
Allowable value	0–59	0–23	1–31	1–12	0–6 Sunday = 0	A valid UNIX command
ערכים מותרים						פקודת UNIX חוקית

ניתן להפריד את שדות הזמן בעזרת רווחים, או בעזרת טאבולטורים. המספרים המייצגים את הזמן יכולים להיות:

* בתחום המצוין בטבלה 12.1.
* כוכבית המציינת את כל הערכים החוקיים.
* רשימת מספרים המופרדים זה מזה בפסיק.
* תחום, המתבסס על שני מספרים המופרדים במקף.

לדוגמה, כדי לבצע את קובץ פקודות המעטפת "seewho.sh", כל עשרים דקות בין יום שני לשישי, יש לכתוב את השורה הבאה בקובץ "crontab":

```
0, 20, 40  *  *  *  *  1-5 seewho.sh
```

בדרך זו יכולה תכנית השירות **cron** לבצע גיבויים יומיים, להריץ תכניות מסוימות בשעות שאין עומס, להגביל הרצת משחקי מחשב בשעות העומס וכן הלאה.

פרק 13

תקשורת בין מערכות UNIX

החלפת מידע בין מערכות UNIX יכולה להתבצע באמצעות הרשת להעתקת UNIX ל-UNIX (להלן uucp - UNIX-to-UNIX copy). אין צורך להיות מומחה בתקשורת כדי לחבר מערכות UNIX בעזרת uucp, אולם, סביר להניח שרשת כזו תותקן בדרך כלל על ידי מנהלן המערכת המקומי.

לשם העתקת נתונים ממערכת אחת לחברתה, יש צורך לשנות את קובצי uucp וקובצי מערכת מסוימים (כמו "passwd"). בידי מנהלן המערכת הרשות לעשות זאת. בעת התקנת רשת uucp, יכול מנהלן המערכת להשתמש בכלים שברשותו כדי למנוע פגיעות אפשריות באבטחה בשימוש ללא הרשאה ברשת.

בפרק זה נעסוק בשיקולים שיש לקחת בחשבון לצורך יצירת קשר בין מערכות UNIX בעזרת uucp. גם נראה, בעזרת דוגמה מעשית, כיצד לקשר בין שתי מערכות.

13.1 התקנת uucp

לפני שמשתמשים ב-uucp במערכת UNIX חדשה יש לבדוק אם רשת uucp הותקנה כבר במערכת. נבדוק אם המדריך "/usr/lib/uucp" קיים ואם הוא קיים - האם נמצאים בו הקבצים "L-devices", "L.sys", "L.cmds", "L.dialcodes", "USERFILE", "uucico", ו-"uuclean". חייבים להיות בו גם המדריכים "/usr/spool/uucp" ו-"/usr/spool/uucppublic".

תהליך ההתקנה של uucp משתנה בין מערכות UNIX שונות. במספר מערכות ניתן להתקין את uucp באמצעות קובץ "uucp.mk" שנמצא ב-"/usr/lib/uucp". עושים זאת בעזרת הפקודה הבאה:

```
$make -f uucp.mk install
```

במערכות אחרות ניתן להתקין את uucp באמצעות תפריט של מנהלן המערכת.

13.2 אפשרויות קישור

יש שלוש אפשרויות לקשר בין מערכות באמצעות uucp:

* חיבור באמצעות רשת חיוג (להלן רשת DDD - Direct Distance Dialling). במקרה זה המעבד משתמש במודם וביחידת חיוג אוטומטית (ACU).

* במרחקים קצרים ניתן לחבר שתי מערכות UNIX בצורה ישירה ללא מודמים. מרחקים מקובלים לחיבור כזה הם כ-270 מ', אך תקן RS232 מציין מרחק של 15 מ' בלבד.

* בעזרת מודם ניתן לחבר שתי מערכות המשתמשות בקו פרטי (נל"ן).

13.3 פקודות uucp

מספר פקודות uucp מאפשרות למשתמש להחליף מידע בין מערכות UNIX. להלן הפקודות:

uucp	מסדרת בתור את הקבצים המיועדים להעברה ומעוררת את "השד" "uucico" במחשב המארח, כדי לבצע את העברת הקבצים למחשב המרוחק. "השד" "uucico" במערכת המרוחקת מופעל כדי לקבל את הקבצים.
uuname	מפיקה רשימה של כל שמות המערכות הידועות ל-uucp.
uux	מבצעת פקודות במערכות מרוחקות.
uulog	שולפת מתוך "usr/spool/uucp/LOGFILE" את הסיכום של תנועות **uucp** ו- **uux**, כדי לקבוע אם הן בוצעו בהצלחה (לדוגמה, **uulog -s**).
uustat	שולפת את מצב סיום ההעברה האחרונה (לדוגמה, **uustat -mall**).
uuclean	מפסיקה את המשימות שעברו על מספר השעות שנקבע להן (לדוגמה "/usr/lib/uuclean -pST -n72").
uuto	העתקת קבצים ציבוריים מ-UNIX ל-UNIX.

המבנה המדויק של הפקודות והאופציות הקיימות בכל אחת מהן נמצאות בתיעוד של המערכת שלך.

13.4 קובצי uucp

כאשר משתמשים ב-uucp יש ליצור, או לשנות מספר קבצים. להלן הסבר על קבצים אלה.

קובץ "/usr/lib/uucp/L.sys"

בקובץ זה ישנה רשימה של המערכות שקבצי uucp יכולים לקרוא להן. המבנה שלו הוא כדלקמן:

```
systemname calltime device bitrate phone expect-and-reply
```

systemname שם המערכת המרוחקת, בקוד ASCII.

calltime המועד שבו יש לקרוא למערכת. המחרוזת "Any" משמעותה שאפשר לקרוא בכל יום, "Wk" משמעותה שאפשר לקרוא לה יום אחד כלשהו בשבוע, המחרוזת "MoFr1000-0800" מציינת שאפשר לקרוא בכל יום בין יום שני לשישי ובכל שעה, פרט לשעות שבין שמונה לעשר בבוקר.

device שם ההתקן (ב-/dev) שמשתמשים בו כדי לקרוא לאותה מערכת. צורת הביטוי היא "ttynn", או הערך "ACU" (יחידת החיוג האוטומטית).

bitrate קצב שידור הנתונים בסיביות לשניה שהמערכת יכולה לטפל בו (לדוגמה, 9600 או 1200 סיביות לשניה).

phone מספר הטלפון של מערכות UNIX שיש להן מתאם לחיוג אוטומטי. מספר זה (לדוגמה, 037321956) צריך להופיע בקובץ "L-dialcodes". במערכות שאין בהן מתאם לחיוג אוטומטי, או במערכות שעובר ביניהן נל"נ, שדה זה צריך להיות זהה לשדה "device".

expect-and-reply "expect" הינה המחרוזת שיש לקרוא מהמערכת המרוחקת ו-"reply" הינה המחרוזת שיש לשדר בחזרה. לדוגמה:

```
ogin-EOT-ogin nuucp
```

שורה זו מודיעה ל-uucp לאתר את המחרוזת "ogin" ואם היא מצאה אותה, עליה לשדר חזרה את המחרוזת "nuucp". אם המחרוזת "ogin" לא נמצאה, יישלח הסימן EOT והתהליך יחזור על עצמו.

שלוש השורות שלהלן הינן דוגמאות של שורות שיכולות להופיע בקובץ "L.sys".

```
brussels  Any,30 ACU 300 206538275  ogin-EOT-ogin: nuucp
host5      Any ttyT16 9600 ttyT16 "" host5 ogin: nuucp
taurus     None Slave 9600 tty12
```

קובץ "L-devices"

קובץ זה מכיל רשימה של כל ההתקנים, אשר מאפשרים חיבור למערכות אחרות. מבנה השורה בקובץ הינו כדלקמן:

```
stringDIR device stringZero bitrate
```

stringDIR	המחרוזת DIR.
device	שמו של ההתקן ב-"/dev". השמות יכולים להיות למשל "ttynn" ו-"ACU". הם צריכים להתאים לערכים הנמצאים בקובץ "L.sys".
stringZero	המספר 0.
bitrate	קצב שידור הנתונים בקו מסוים (סיביות לשניה, למשל 9600).

לדוגמה:

```
DIR tty05 0 9600
```

שורה זו מודיעה ל-uucp ש-"/dev/tty05" הינו קו נל"ן וקצב שידור הנתונים בו 9600 סיביות לשניה.

קובץ "SYSTEMNAME"

קובץ זה (ניתן למצוא אותו רק במספר מערכות UNIX) מכיל את שם המערכת המארחת (לדוגמה S8500).

קובץ "L.cmds"

קובץ זה מכיל פקודות UNIX, אשר תוכנת "uuxqt" תרשה לבצע אותן במערכת המרוחקת. בכל שורה יש פקודה אחת כפי שנראה להלן:

```
cp
lp
ps
ls
uucp
```

קובץ "L.dialcodes"

קובץ זה משמש מערכות UNIX, שבהן יש חייגן אוטומטי. באמצעותו ניתן למפות את קודי החיוג ומספרי הטלפון ולשייכם זה לזה.

קובץ "USERFILE"

קובץ זה מכיל מידע, שאליו יכולים לגשת הן המשתמש והן המכונה. מבנה הרשומה בקובץ הוא כדלקמן:

```
[loginame],[systname] [c] path [path]
```

loginame	השם שבו המשתמש נכנס למערכת, או המחשב המרוחק.
systname	שם המערכת של המחשב המרוחק.
c	דגל (flag) לציון חיוג חוזר (call-back); כאשר הוא ישנו, יתקשר המחשב המארח למחשב המרוחק כדי לאשר את זהותו.
path	שם מסלול מקובל.

כדוגמה, התבונן בשורות הבאות מתוך הקובץ USERFILE:

```
andy,S8500 /usr
, /usr/work
, /work
```

השורה הראשונה מצביעה על כך שהמערכת "S8500" יכולה להכנס (login) תחת השם "andy" ולהעביר קבצים אל המדריך "/usr", או ממנו. השורה השניה מציינת שכל משתמש יוכל להעביר קבצים דרך המדריך "/usr/work". ולבסוף, השורה השלישית מצביעה על כך שכל משתמש יכול להעביר כל קובץ הפתוח לכולם.

13.5 הקבצים "password" ו-"inittab"

שני קובצי מערכת צריכים לעבור שינוי בזמן שמתקינים את uucp: "/etc/passwd" ו-"/etc/inittab".

קובץ "/etc/passwd" הינו קובץ הסיסמאות של UNIX, אשר מתאר כל משתמש במערכת. יש ליצור רשומות בקובץ הסיסמאות לכל אחד ממנויי uucp, כדי לאפשר למערכות רחוקות להתקשר למערכת המארחת. להלן רשומה טיפוסית שיש ליצור:

```
nuucp::6:1:0000-uucp(0000):/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

יש לשים לב לכך שברשומה זו המדריך "uucppublic" משמש כמדריך הפעיל של uucp והשד "uucico" משמש לצרכי המעטפת.

המנהלן צריך גם ליצור רשומה בקובץ הסיסמאות שתאפשר לו להתערב בפעולת uucp. הנה רשומה טיפוסית:

```
uucp:AnPqy:5:1:0000-uucp(0000):/usr/lib/uucp:/bin/sh
```

יש ליצור את הרשומות שתוארו לעיל הן במערכת המארחת והן במערכת המרוחקת.

יש לשנות גם את הקובץ "/etc/inittab". קובץ זה משמש את "/etc/init", כדי לקבוע אילו תהליכים יווצרו ואילו תהליכים יסתיימו בכל מצב של init. הרשומה בקובץ "inittab" היא בעלת המבנה הבא:

```
id:initlevel:type:processaction
```

הבה נתבונן במקרה שבו רוצים ליצור חיבור uucp ישיר בין "tty05" במערכת המארחת לבין-"tty06" במערכת הרחוקה (ראה גם תרשים 13.1). במקרה זה יש לשנות את השדה "type" במערכת המארחת ל-"off" ובמערכת הרחוקה יש לשנות שדה זה ל-"respawn". לאחר ביצוע שינויים אלה יש לבצע את הפקודה:

```
$/etc/init q
```

פקודה זו תאלץ את "/etc/init" לקרוא מחדש את הקובץ "/etc/inittab" מבלי לשנות את מצבי init. דבר זה מונע את הצורך לאתחל (reboot) את המערכת.

13.6 הרשאות הכניסה לקבצים מיוחדים

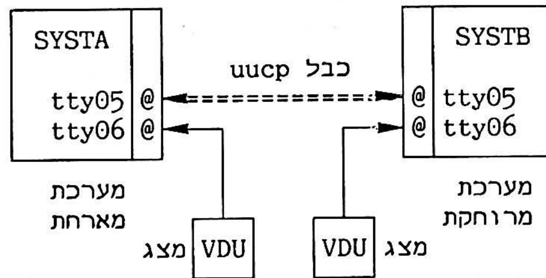
כדי שהעברת הקבצים תצליח, יש צורך לשנות את הרשאות הכניסה ל-"666" בקבצים הנמצאים במדריך "/dev" ומעורבים בתהליך uucp. אם, לדוגמה, משתמשים ב-"tty05"" הן במערכת המארחת והן במערכת המרוחקת, יש לשנות את הרשאות הכניסה לקבצים המיוחדים בעזרת הפקודה הבאה:

```
$chmod 666 /dev/tty05
```

13.7 דוגמה מעשית

תרשים 13.1 מציג חיבור ישיר על ידי הצלבה (cross-coupling) של שתי מערכות UNIX. מערכות אלו הן "SYSTA" ו-"SYSTB".

לצורך הדוגמה, תפיק הפקודה "$uuname -1" את הפלט של SYSTA במערכת המארחת ואת הפלט SYSTB במערכת המרוחקת. בדוגמה זו תשמש הכניסה (port) בשם tty05 לחיבור בין שתי המערכות. SYSTA תהיה המערכת הראשית (master) והמערכת SYSTB תהיה כפופה לה (slave). בהמשך נתאר את הרשומות שיש ליצור בקובצי ה-uucp בשתי המערכות.



תרשים 13.1 חיבור על ידי הצלבה (בלי מודם)

קובצי המערכת המארחת (SYSTA)

הסימן # בקובץ שלהלן משמש לציון שורת הערה.

```
#The L.sys file contains

SYSTB Any tty05 9600 tty05 "" ^M ogin:-^M-ogin:-^M-ogin:nuucp

#The L-devices file contains
DIR tty05 0 9600


#The L.cmds file contains
rmail
uucp
ls
who
lp
```

```
#The USERFILE file contains
nuucp, /usr/spool/uucppublic /usr/spool/uucp /usr/spool/mail
, /


#The tty05 entry in the /etc/inittab file is
A5:1:off:/etc/getty tty05 9600


#The /etc/passwd file entries for uucp and nuucp are
uucp:AnPqy:5:1:0000-uucp(0000):/usr/lib/uucp:
nuucp::6:1:0000-uucp(0000):/usr/spool/uucppublic:/usr/lib/uucp/uucico

#The mode for /dev/tty05 is 666
-rw-rw-rw- --- - - - - - - - - - - - - tty05
```

קובצי המערכת המרוחקת (SYSTB)

```
#The L.sys file contains
SYSTA Any Slave 9600 tty05


#The L.cmds file is the same as on SYSTA.


#The USERFILE file is the same as on SYSTA.


#The tty05 entry in the /etc/inittab file is
A5:1:respawn:/etc/getty tty05 9600


#The /etc/passwd file entries for uucp and nuucp are
uucp:zy4Ac:5:1:0000-uucp(0000):/usr/lib/uucp:
nuucp::6:1:0000-uucp(0000):/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

13.8 העברת קבצים

ניתן להשתמש בפקודה uucp להעברת קבצים בין שתי מערכות UNIX או יותר. מבנה הפקודה:

```
$uucp [options] source destination
```

השדות "source" ו-"destination" יכולים להיות שם מסלול, או

שהם יכולים להיות במבנה הבא:

```
systemname!pathname
```

"systemname" הינו שם מערכת שהפקודה uucp מכירה.

לדוגמה, כדי להעביר קבצים בין SYSTA לבין SYSTB לפי הדוגמה בסעיף הקודם, צריך להכנס תחילה ל-SYSTA. לאחר מכן נשתמש בפקודה הבאה כדי להעביר את הקובץ "myfile" מהמדריך John למדריך uucppublic שב-SYSTB:

```
$uucp -j /usr/acct/john/myfile SYSTB!/usr/spool/uucppublic
```

באופן דומה, ניתן להעביר קובץ מ-SYSTB ל-SYSTA כאשר נמצאים ופועלים ב-SYSTA. נשתמש בפקודה הבאה:

```
$uucp -j SYSTB!/usr/spool/uucppublic/myfile SYSTA!/usr/uucppublic
```

לכל בקשת uucp קשור מספר עבודה (job number). ניתן להציג מספר זה בעזרת האופציה j- (ראה הדוגמה לעיל). אפשרות אחרת היא לכלול את השורה הבאה בקובץ "$HOME/.profile".

```
JOBNO=ON; export JOBNO
```

אם המשתנה "JOBNO" שווה ל-"OFF", או שהוא אינו מוגדר, לא יוצג מספר העבודה.

13.9 בעיות וניפוי שגיאות

מספר בעיות יכולות להיות בזמן שמשתמשים ב-uucp:

* יחידות החיוג האוטומטיות מחייגות בלי הפסק בניסיון ליצור קשר עם מערכות מרוחקות, שמספרי הטלפון או הסיסמאות שלהן השתנו.
* לא נותר מקום אחסון במערכות המקושרות.
* תקלות במודם, או בקווי התקשורת.
* שיבושים בקבצי uucp, או מבנה לא נכון של קובץ uucp.
* קובצי uucp והתוכנות אינם מכילים את מספר הזיהוי (ID) הפרטי או הקבוצתי הנכונים, סיבית זיהוי או הרשאות כניסה (לדוגמה, בתכנית "uucico" יש לקבוע את סיבית הזיהוי של המשתמש - SUID);

אם יש חשד לקו לא תקין, ניתן להשתמש בפקודת **cu** (ראה להלן) כדי לזהות את הבעייה.

הבה נתבונן בחיבור מהסוג המתואר בדוגמה המעשית שהוצגה בסעיפים קודמים. ניתן להשתמש בפקודת **cu** כדי לבדוק אם ניתן לקרוא ל-SYSTB מתוך SYSTA. אפשרות אחרת היא להשתמש בשתי הפקודות הבאות:

```
$uucp -r myfile SYSTB!/usr/spool/uucppublic/myfile
$/usr/lib/uucico  -rl -x4  -sSYSTB
```

myfile	הקובץ המיועד להעברה מהמדריך הנוכחי, (אל SYSTB, בדוגמה שלנו).
-r	מציין שיש להכניס לתור את העבודה (job) של uucp מבלי להפעיל את השד "uucico".
-rl	מציין שהשד "uucico" צריך להתחיל ב-master mode.
-x4	מציין את רמת ניפוי השגיאות.

בעזרת הפקודות הנ"ל ובעזרת הפקודות הבאות ניתן לזהות בדרך כלל את הבעיות הקשורות ל-**uucp**:

```
$uustat -mall
$uulog -s<system-name>
```

13.10 אחזור קבצים שהתקבלו

קבצים הנשלחים למשתמש מסוים באמצעות הפקודות **uucp** ו-**uuto** מאוחסנים בדרך כלל במדריך

/usr/spool/uucppublic/receive/username

ניתן להעביר קבצים אלה למדריך של המשתמש בעזרת הפקודה **mv**, או בעזרת הפקודה **uupick** שהתחביר שלה הוא:

```
$uupick [-s systemname]
```

עבור כל קובץ שהתקבל ונרשם במדריך המיועד לאותו משתמש, תדפיס **uupick** את ההודעה הבאה בפלט הסטנדרטי:

<u>from</u> systemname : [<u>file</u> filename] [<u>dir</u> dirname]?

הערה: המלים המסומנות בקו הינן מלים קבועות.

המשתמש יכול לענות לאחר מכן באחת מהתשובות הבאות:

m[dir]	העבר את הקובץ למדריך המצוין בפקודה (המדריך הנוכחי הינו ברירת המחדל).
newline	הבא את הקובץ הבא.
d	בטל את הקובץ.
q	עצור.
*	הדפס את כל הפקודות האפשריות.

13.11 קריאה למערכת אחרת בעזרת cu

ניתן להשתמש בפקודה **cu** לקריאה למסוף אחר, לקריאה למערכת UNIX אחרת ובמספר מקרים - למערכת שאינה UNIX. ניתן לבצע פקודות במערכת מקומית ובמערכת מרוחקת המחוברות באמצעות **cu** וניתן להעביר קבצים בין שתי המערכות.

cu ככלי לניפוי שגיאות

ניתן להשתמש ב-**cu** כדי לבדוק אם יש בעיות בהתקנה של "uucp". אם לא ניתן לקרוא למערכת המרוחקת, יכולה הסיבה להיות כבל לא נכון, או חיבור גרוע. אי לכך, אם נשתמש ב-**cu** לאחר התקנת "uucp" נמנע בעיות אלו.

כדוגמה, הבה נתבונן בחיבור בין SYSTA לבין SYSTB שהודגם בסעיפים קודמים. אם נקיש את הפקודה הבאה

```
$cu -d -l/dev/tty05
```

תהיה לנו אפשרות להכנס מ-SYSTA אל SYSTB.

מסך הכניסה של SYSTB צריך להופיע במסוף של SYSTA. אם אין זה קורה, סימן שהחיבור פגום, או שהכבל לא תקין. הדגל "d-" מדפיס נתוני מעקב לצורכי דיאגנוסטיקה כדי לעזור בתהליך ניפוי השגיאות. הדגל "l-" מציין את ההתקן שיש להשתמש בו כערוץ תקשורת.

פקודות cu

לאחר שהוקם הקשר בין המערכות בעזרת פקודת **cu**, כפי שראינו קודם, עומדות לרשותנו במספר פקודות כדי להעביר ביניהן קבצים. בעזרת הפקודה הבאה ניתן להעביר את הקובץ "master"

מ-SYSTA ל-SYSTB:

```
~%put/work/andy/master /usr/andy/master
```

באופן דומה תעביר הפקודה הבאה את הקובץ "test" מ-SYSTB ל-SYSTA:

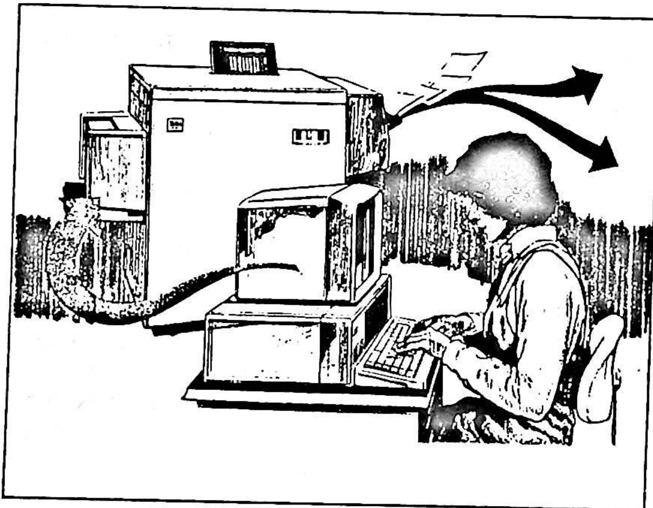
```
~%take/work/andy/test /usr/andy/test
```

אם נכתוב כך,

```
~$com
```

הפקודה **com** תתבצע במערכת שבה ניתנה ותשלח את הפלט שלה למערכת המרוחקת. ולבסוף, כדי להפסיק את התקשורת באמצעות **cu**, השתמש בפקודה הבאה:

(שים לב: התו ~ ונקודה מימינו)



פרק 14

פונקציות מנהלה כלליות

בפרק זה נתרכז בהיבטים הקשורים למערכת ההפעלה UNIX, אשר נמצאים באחריותו של מנהלן המערכת. נמצא בו גם מידע שימושי למשתמש UNIX רגיל, מכיון שגם הוא עוסק בנושאים אלה:

* תיחול (booting) המערכת.
* רצף הפקודות **init-getty-login-shell**.
* גיבוי מלא וגיבוי חלקי (ארכיב).
* הוספה וביטול של משתמשים.
* ניצול קיבולת הדיסק (**df,du**).
* מבנה מערכת הקבצים.
* יצירת מערכת קבצים.
* תחזוקת מערכת הקבצים (**fsck ,ncheck**).
* ניקוי חירום של מערכת הקבצים.
* הוספת התקנים חדשים למערכת UNIX.

14.1 תהליך התיחול

הנוהל הדרוש להפעלת מערכת UNIX שונה אצל יצרני חומרה שונים. אולם התהליך הכללי והקבצים שמעורבים בהפעלת המערכת כמערכת רבת משתמשים, או כמערכת של משתמש יחיד, הינם דומים.

תהליך התיחול מתחיל לאחר הדלקת המערכת. בתהליך זה יוזמת החומרה את הטעינה לזיכרון של תכנית טעינה קטנה (bootstrap loader program). תכנית זו נמצאת בבלוק הראשון של הדיסק, או בשטח המיועד למטען המערכת (ראה פרק 2).

תכנית הטעינה טוענת לזיכרון את הקובץ "unix/", אשר מכיל את גרעין מערכת UNIX. תהליך התיחול ממשיך להפעיל את ממשקי החומרה, מכין לעבודה את מאגרי הבלוקים, את המאגרים של inode, את המשתנים של הזיכרון המרכזי ואת השעון שמספק פסקים בתדירות קבועה ומבצע גם הוא אתחולים אחרים.

לאחר פעולות אלו, הגרעין יוצר ומאתחל מבנה נתונים הידוע כתהליך 0 (אפס). תהליך זה אינו כולל איזור פקודות ותפקידו לתמוך בגרעין לניהול תהליכים ולתזמון פעולות. תהליך 0 ימשיך

להתקיים כל עוד המערכת פועלת.

תהליך 1 הינו צאצא של תהליך 0. מוקצה לו זיכרון ונטענת בו תכנית לתוך איזור הפקודות. קריאת **exec** בתכנית מבצעת את התכנית "/etc/init", אשר תופסת את מקומן של הפקודות המקוריות שבתהליך 1. תהליך 1 הופך מנקודה זו ואילך לתהליך התיחול (init process).

14.1 רצף הפקודות init-getty-login-shell

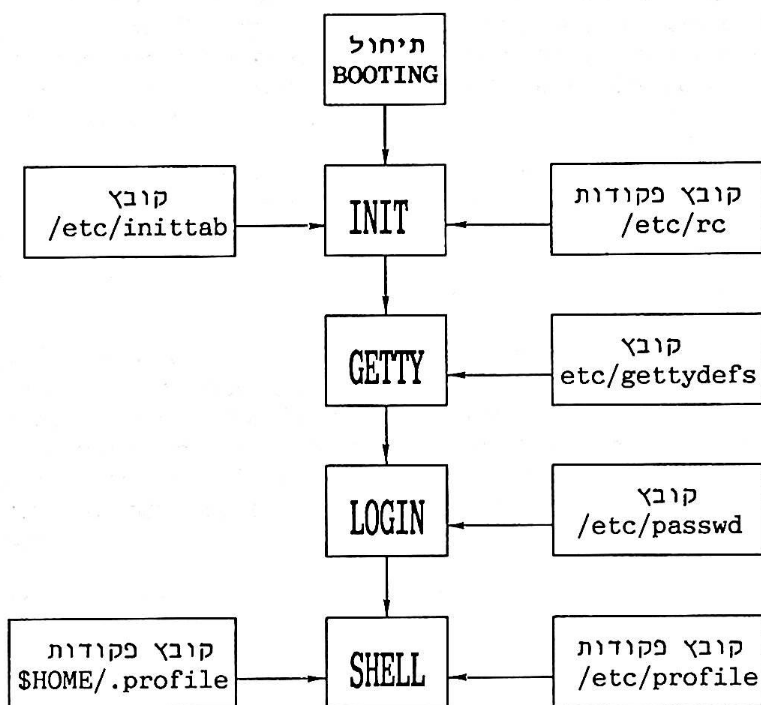
לאחר תהליך התיחול שתואר לעיל, אחראית **init** לקביעת שתי סביבות עיבוד שונות; סביבת משתמש יחיד וסביבת עבודה רבת-משתמשים.

הפקודה **init** תיצור מעטפת עם הרשאות של משתמש-על ותקצה אותה לקונסול של המערכת. באופן הפעלה זה (משתמש יחיד), יכול מנהלן המערכת לבצע תחזוקה, פעולות שינוי וגיבוי של קבצים. בנקודה זו מקובל גם להשתמש בפקודת **/etc/fsck** כדי לבדוק מערכות קבצים פגומות ולתקן אותן בצורה הדברותית (ראה fsck).

כדי לגרום לכך שהמערכת תהיה רבת-משתמשים, יש להשתמש בפקודה **init 2**, או להכנס למערכת תחת שם מיוחד (כדאי שתבדוק את אפשרויות המערכת שלך בענין זה). בזמן שעוברים ממצב של משתמש יחיד למצב של משתמשים רבים, יוצרת **init** מעטפת למשתמש-על, אשר מבצעת את הפקודות הנמצאות בקובץ "/etc/rc". הביצוע של "/etc/rc" מבוקר על ידי שורה בקובץ "/etc/inittab" שכלול בה מידע לניתוב העיבוד עבור הפקודה **init**.

הפקודות הנמצאות ב-"/etc/rc" משתנות ממערכת UNIX אחת לרעותה. הן משמשות לניקוי קבצים כמו "/etc/utmp", להתקנת מערכות קבצים, לביטול קבצים במדריך "/tmp", לניקוי הקבצים הנעולים ב-spooler, להצבת תאריך במערכת, לאפשר חיוב עבור עבודות (accounting) ועוד.

תפקיד נוסף וחשוב של **init** הוא ליצור תהליכים, אשר יהוו חלק מרצף הפקודות **init-getty-login-shell** שמאפשר למשתמשים להכנס למערכת (ראה תרשים 14.1). לרוב, **init** מבצעת את **getty** מתוך שורה בקובץ "/etc/inittab". הפלט של התכנית "getty" מכיל את מספר הזיהוי של המערכת, תכולת הקובץ "/etc/issue" והודעת הכניסה במסוף. כאשר מישהו מנסה להכנס למערכת, קוראת "getty" את השם שהוקש ומפעילה את פקודת **login** עם שם המשתמש כארגומנט. באותו זמן קובעת **getty** את מאפייני המסוף ואת קצב שידור הנתונים על פי המידע שנמצא בקובץ "/etc/gettydefs".



תרשים 14.1 תיחול המערכת

הפקודה **init** משעה את פעולתה כאשר המערכת תומכת במשתמשים רבים ומחדשת אותה כתוצאה מתיחול, אות של המשתמש, תקלה באספקת המתח למערכת, או כתוצאה מ"מותו" של אחד מבניה.

בכל המקרים האלה, היא תסרוק את הקובץ "/etc/inittab" ותנקוט בפעולה המתאימה. אם לדוגמה "מת" תהליך המפקח על מסוף, **init** תיצור **getty** חדש בעזרת **fork/exec**, כדי לשלוט בקו ותמתין עד אשר המשתמש הבא ייכנס למערכת.

14.3 כיבוי מערכת UNIX

מדי פעם צריך ל"סגור" את מערכת UNIX. ניתן לעשות זאת על ידי כך שנכנסים אליה כמשתמש-על ושולחים ל-init את אות הסיום SIGHUP בעזרת הפקודה הבאה:

```
$kill -1 1
```

המשתמש צריך לבדוק אם במערכת שלו צריך לשלוח את "אות 1" (SIGHUP) אל "תהליך 1" ("init") כדי לסגור את המערכת. פקודה **kill** כזאת תפסיק את פעולתם של התהליכים שמתבצעים באותו רגע ותחזיר את המערכת למצב של משתמש יחיד עם מעטפת שפועלת על מסוף הקונסול של המערכת בלבד.

כדי לצמצם למינימום את התנועה לדיסק וממנו, שומרת UNIX מידע רב ככל האפשר במאגרים של הזיכרון המרכזי. לפני שמבצעים פונקציות מנהלה כלשהן במצב של משתמש יחיד, חשוב להקיש את הפקודה הבאה:

```
$sync
```

הפקודה **sync** מנקה את המאגרים של הזיכרון המרכזי ומשאירה את מערכת הקבצים ללא שינוי. במספר מערכות UNIX ישנו קובץ פקודות מעטפת הנקרא "shutdown" שמיועד לכבות את המערכת בצורה מדורגת. לדוגמה, הפקודה הבאה תעביר את המערכת למצב של משתמש יחיד תוך 180 שניות, לאחר שתשלח הודעות אזהרה לכל המשתמשים הפעילים. מנהלן המערכת יוכל אז לבצע את הדברים הבאים:

* גיבויים יומיים.
* כיבוי המחשב.
* תחזוקת המערכת (ראה **fsck**).
* החזרה של המערכת למצב של תמיכה במשתמשים רבים.

14.4 גיבוי מלא לעומת גיבוי חלקי

גיבוי מלא וגיבוי חלקי הן שתי דרכי פעולה שנועדו להגן על המשתמש בפני פגיעה או תקלה במערכת הקבצים. בגיבוי מלא נעשה עותק מושלם של המערכת לאמצעי הגיבוי, כמו סרט מגנטי. זוהי שיטה פשוטה יחסית ואמינה.

בגיבוי חלקי יש יותר עבודה מכיון שקבצים נבחרים בלבד מועתקים לסרט או לתקליטון. כדאי לגבות קבצים גדולים שאינם נמצאים בשימוש תדיר ולבטל אותם מן הדיסק. בכל פעם שיש צורך בהם, ניתן לשחזר אותם מחדש על ידי טעינה חוזרת לדיסק.

יש לבצע גיבוי מלא על בסיס יומי או בטווחי זמן אחרים, בהתאם לרמת הפעילות והשינוי בקבצים ובתכניות. כך ניתן להתגבר על תקלות כתוצאה מנפילות של המערכת וטעויות של המשתמש. בסעיף זה נתאר שלוש פקודות שמשמשות לביצוע גיבויים במערכת UNIX: **cpio**, **tar** ו-**dd**.

הפקודה cpio

המלה **cpio** הינה קיצור של **'copy file archive in and out'**, כלומר, גיבוי של קובץ ושחזורו. לפקודה **cpio** יש מספר אופציות (ראה בדוגמאות):

o- קרא רשימה של שמות מסלולים מהקלט הסטנדרטי והעתק את הקבצים האלה לפלט הסטנדרטי (כלומר, גיבוי קובץ - copy out).

i- קרא שמות קבצים מהקלט הסטנדרטי (כלומר, copy in).

d צור מדריכים לפי הצורך.

v פרט את שמות הקבצים שעומדים להיות מודפסים (כלומר, מצב מרובה מידע).

u החלף קובץ ללא תנאי.

c כתוב כותרת המכילה מידע בשיטת ASCII, כדי שאפשר יהיה לנייד את הקובץ.

B בצע קלט/פלט ביחידות בנות 5120 בתים לרשומה.

t הדפס טבלה המפרטת את תכולת הקלט.

שלושת קובצי פקודות המעטפת שלהלן מראים כיצד ניתן להשתמש ב-**cpio** לביצוע גיבוי של קבצים במדריך משתמש מסוים (John), כיצד להציג את הקבצים שגיבנו וכיצד לשחזר קובץ כלשהו רצוי.

```
#
#Script-1 to backup the directory /usr/john
#
echo "Backing up the directory /usr/john\n"
echo "cd /usr/john"
cd /usr/john
find . -print | cpio  -ovcB  > /dev/rtp
echo "\nBackup finished \n"
#
#Script-2 for showing what was backed-up on a tape
#
echo "Show what was backed-up \n"
cpio -iBtvc < /dev/rtp
echo "Contents of entire tape have been shown. Have fun now !\n"

#Script-3 for individual file recovery
#
echo "cd /usr/john\n"
 cd /usr/john
cpio -iuvBc $1 < /dev/rtp
echo "Recovery of $1 finished\n"
```

אם "script-3" היה נקרא "indvidual" והמשתמש היה רוצה לשחזר קובץ הנקרא "swap.c" ממדריך "/usr/john/projecta", הוא היה צריך להשתמש בפקודה הבאה:

```
$indvidual projecta/swap.c
```

הפקודה tar

הפקודה **tar** (tape archiver) מקובלת לביצוע פעולות ארכיון. היא מעתיקה קבצים ממדריך מסוים אל אמצעי גיבוי כמו סרטים מגנטיים או תקליטונים. המידע המשמש לצורכי בקרה נכתב ב-ASCII, דבר העוזר לניידות של אמצעי הגיבוי. להלן מספר אופציות של פקודת **tar**:

c הכן סרט tar חדש לגיבוי. התחל לכתוב מתחילת הסרט.
r רשום את שמות הקבצים בסוף הסרט. ניתן להשתמש באופציה זו לגיבוי חלקי, שבו כותבים רק את הקבצים שהשתנו מאז הגיבוי האחרון.
v הצג מידע על הקבצים שגובו.
b השתמש בארגומנט הבא לצורך קביעת מספר הרשומה בבלוק (blocking factor).
f השתמש בארגומנט הבא כשם הארכיב. אם שם הקובץ הוא -, תקרא **tar** מהקלט הסטנדרטי, או תכתוב לפלט הסטנדרטי, מותנה באופציות האחרות.
x שלוף קבצים מסוימים מתוך אמצעי הגיבוי.

לדוגמה, כדי לגבות את הקבצים הנמצאים במדריך "/usr/john", ובהנחה ששם ההתקן הינו "/dev/rtp", יש להשתמש בפקודה הבאה:

```
$tar cvbf 10 /dev/rtp /usr/john
```

ניתן לגבות קבצים של מספר חשבונות משתמשים (accounts). נשתמש בתכנית העריכה כדי ליצור, לדוגמה, במדריך "/usr" את הקובץ "allacc", שמכיל את מספרי החשבונות שיש לגבות, כמו למשל,

```
/usr/andy
/usr/john
/usr/peter
```

לאחר מכן ניתן לגבות את החשבונות שב-"allacc" בעזרת הפקודה הבאה:

```
$tar cvbf 10 /dev/rtp 'cat /usr/allacc'
```

כדי לשחזר את המדריך "/usr/john", בהנחה שפקודה **tar** הנ"ל

שימשה לגיבוי, יש להשתמש בפקודה הבאה:

```
$tar xvbf 10 /dev/rtp /usr/john
```

הפקודה dd

ניתן להשתמש גם בפקודה **dd** לביצוע גיבויים. היא מבצעת העתקה ישירה מהתקן להתקן ומכיון שהיא מאפשרת לכתוב ולקרוא בגודל בלוק שרירותי כלשהו, היא משמשת בעיקר לקלט/פלט של התקנים פיסיים שלמים.

הפקודה **dd** תשמש בדרך כלל להעתקת דיסק שלם, מסילה אחר מסילה, לסרט מגנטי. מבנה הפקודה:

```
$dd if=/dev/disk2 of=/dev/rmt1 [options]
```

מקרא:

/dev/disk2	שם של הדיסק הקשיח המשמש לקלט.
/dev/rmt1	שם הסרט שאליו יועתק הדיסק.

המלים "if" ו-"of" הינם קיצורים של "input file" (קובץ קלט) ו-"output file" (קובץ פלט) בהתאמה. להלן כמה מבין האופציות של הפקודה.

ibs=n	גודל בלוק הקלט הוא **n** בתים.
obs=n	גודל בלוק הפלט הוא **n** בתים.
bs=n	גודל בלוק הקלט וגודל בלוק הפלט הוא **n** בתים.
count=n	העתק מהקלט **n** בלוקים בלבד.
conv=ebcdic	המר ASCII ל-EBCDIC.
conv=ascii	המר EBCDIC ל-ASCII.
conv=swab	החלף כל זוג של בתים.

שים לב! גודל בלוק הקלט וגודל בלוק הפלט תלויים בסוג האמצעי המשמש לגיבוי. הדוגמאות שלהלן מראות כיצד ניתן לשלב את הפקודות **dd** ו-**cpio** לביצוע גיבוי בצורה יעילה יותר.

לגיבוי ניתן להשתמש בפקודה הבאה:

```
ls * ¦ cpio -ocv ¦ dd of=/dev/rtp obs=100k
```

לאחר מכן, אפשר להשתמש בפקודה הבאה כדי לשחזר את הקבצים:

```
$dd if=/dev/rtp bs=100k ¦ cpio -icvdum
```

המשתמש ב-**dd** צריך להיות מודע לכך שבמקרה שיש בדיסק בלוקים פגומים, יכול הדבר לגרום להפסקת ביצוע הפקודה, או לגרום לתקלות בזמן שחזור הקבצים לדיסק אחר. מכאן שהפקודה **dd** מתאימה לגיבוי ושיחזור באותו דיסק, כאשר תוך כדי פעולות אלו יש לנטרל (unmount) את מערכת הקבצים כדי לשמור על עקביות במצב הדיסק.

14.5 הוספת משתמשים חדשים

הנוהל להוספת משתמשים חדשים יכול להתבצע באופן אוטומטי באמצעות קובץ פקודות מעטפת (shell script). לקובץ פקודות כזה אין שם סטנדרטי. במסגרת התהליך הבסיסי דרושות מספר פעולות.

* יש להחליט על מספר מזהה ייחודי למשתמש (UID). ניתן לבחון את הייחודיות בעזרת הפקודה הבאה:

```
$grep ":.*:.*:UID-number" /etc/passwd
```

לא יודפס דבר אם ה-UID שבחרנו הינו ייחודי במערכת.

* המספר המזהה של הקבוצה (GID) שאליה שייך המשתמש. GID הינו מספר קיים וההחלטה תהיה קלה יותר לאחר בדיקה של הקובץ "/etc/group". אם המשתמש יורשה לשנות את קבוצת השייכות שלו באמצעות הפקודה "newgrp name-of-group", יש לכלול את שם המשתמש בכל אחת מהקבוצות שאליהן הוא יהיה רשאי להסתפח.
* סיסמה ייחודית לצורך כניסה למערכת; הייחודיות יכולה להבדק באמצעות הפקודה הבאה:

```
$grep "^user-name:" /etc/passwd
```

* המעטפת שהמשתמש יבחר להשתמש בה (מעטפת בורן, מעטפת C, או מעטפת מוגבלת).
* יצירת מדריך הבית למשתמש חדש.

להלן קובץ פקודות מעטפת להוספת משתמשים חדשים למערכת:

```
# Script        : adduser
#
# Input argms   : $1 is the name of the user to be added to the system
#

# remember that   $0  is the name of the the script
```

```
if [ "$#" != 1 ]; then
 echo "Usage: $0 new-user-name\n"; exit 1
fi

echo "$0:Checking to see if another user exists with the \
same login-id\n"
userid=$1
if grep "^$userid:" /etc/passwd > /dev/null ; then
    echo "$0: Another user with name \"$userid\" already\
exists\n"
    exit 1
fi

echo "$0:Creating a unique uid\n"
uid='cut -f3 -d: /etc/passwd|sort -n|tail -1'
uid='expr $uid + 1'
#if uid is less than or equal to 100 make uid equal to a 100
test $uid -le 100 && uid=100

echo "$0:Writing user entry to the password file\n"
echo "$userid::$uid:100:$userid:/usr/$userid:/bin/sh"  >>/etc/ passwd

echo "$0:Creating user's directory and changing permissions\
appropriately\n"
mkdir /usr/$userid
chown $userid /usr/$userid
chgrp usr /usr/$userid
chmod 0744 /usr/$userid
echo "$0:New user has now been added to the system\n"
echo "$0:You may now give to the user his/her own\
\$HOME.profile file\n"
```

14.6 ביטול משתמשים

ביטול משתמשים במערכת יכול להעשות בעזרת קובץ פקודות מעטפת המופעל לרוב מתפריט. יש לנקוט את הצעדים הבאים:

* לבטל את רשומת המשתמש בקובץ "/etc/passwd".
* לבטל כל התייחסות למשתמש בקובץ "/etc/group".
* להשתמש בפקודה **rm -rf** מתוך מדריך הבית של המשתמש, כדי לבטל את כל הקבצים וכל תת-המדריכים הנמצאים באותו מדריך.

להלן קובץ פקודות מעטפת טיפוסי לביטול משתמשים במערכת UNIX.

```
# Script           : duser
#
# Input parameters : $1 specifies the user to be deleted
#                    $2 specifies user's directory to be deleted
#

theusr=$1
out=/tmp/userd$$

echo "$0:Checking to see if user's directory is to be deleted\n"
if [ -n "$2" ]
 then
    dirdel=true
    dir=$2
else
   dirdel=false
fi

echo "$0:Checking to see if user can be deleted\n"
case $theusr in
     root | uucp | sync ) echo "$0:You can not delete root,uucp\
                                 or sync users\n"
                          exit 1;;

esac

echo "$0:Checking to see if user exists\n"
grep "^$theusr:" /etc/passwd > /dev/null
case $? in
        0 ) ;;

        * ) echo "$0:$theusr cannot be located\n"
            exit 1;;
esac

#use the "sed" utility to delete the user entry
echo "$0:Deleting the user from the password file\n"
sed -e "/^$theusr:/d" /etc/passwd > $out
cp $out /etc/passwd
rm $out
```

```
echo "$0:Deleting user from group file\n"
ed - /etc/group <<%
g/:$usr$/s//:/
g/:$usr,/s//:/
g/,$usr$/s///
g/,$usr,/s//,/
w
q
%

if [ $dirdel ] ; then
       echo "$0:Removing user's directory\n"
       rm -rf $dir
fi

echo "$0:User $theusr removed from the system\n"
exit 0
```

14.7 ניצול קיבולת הדיסק

מעקב אחר קיבולת הדיסק התפושה והפנויה, הינו תפקידו של מנהלן המערכת. במערכות UNIX קטנות עושים זאת משתמשים רגילים. אם נגמר המקום למערכת קבצים כלשהי, יתחילו מספר תכניות שירות (לדוגמה **uucp**) וחבילות יישומים המשתמשות בה לתת הודעות שגיאה לא נכונות. אם מספר צמתי i (inodes) הינו קטן, יבוזבז הרבה מזמנו של ה-CPU לבניית מערך חופשי של inodes. ניתן לגלות ולמנוע בעיות אלו אם משתמשים באופן סדיר במספר פקודות, אשר מספקות מידע מועיל על ניצול הדיסק. שתיים מפקודות אלו הן **df** (השטח הפנוי בדיסק) ו-**du** (השימוש בדיסק).

הפקודה df

הפקודה **df** מציגה את מספר הבלוקים הפנויים בכל מערכות הקבצים או במערכת קבצים מסוימת. המבנה הבסיסי של הפקודה:

$df [-אופציות] [מערכת קבצים]

אופציות:

-f להציג את מספר הבלוקים הפנויים.
-t להציג את כל הבלוקים שהוקצו, ואת מספר הבלוקים שהוקצו לכל מערכת קבצים.

פלט טיפוסי של הפקודה **df** ושל הפקודה **df -t**:

```
$df
/usr/acct (/dev/dsk/1s1):  39216 blocks 7095 i-nodes
/         (/dev/dsk/0s1):  11984 blocks 5017 i-nodes

$df -t
/usr/acct (/dev/dsk/1s1):  39216 blocks 7095 i-nodes
                   total:  68186 blocks 8512 i-nodes
/         (/dev/dsk/0s1):  11984 blocks 5017 i-nodes
                   total:  68186 blocks 8512 i-nodes
```

הפקודה du

הפקודה **du** מציגה את ניצול הדיסק, במונחים של בלוקים, לפי מדריך מסוים, או לפי המדריך הנוכחי, אם לא צויין שם מדריך בפקודה.

$du [-אופציות] [שם מדריך]

אופציות:

-s להציג את הסיכום הכללי (grand total).
-a להציג סיכום לכל קובץ.

פלט טיפוסי של הפקודה **du**:

```
$du /usr | sort -n
434 /usr/andy/accounts
445 /usr/andy/databases
883 /usr/andy
```

14.8 מבנה מערכת הקבצים ב-UNIX

בתרשים 14.2 מוצג מבנה של מערכת קבצים ב-UNIX (ראה גם את הפקודה **mkfs**).

הבלוקים יכולים להיות בגודל של 512 בתים (גרסאות UNIX מוקדמות), או 1024 בתים. פקודות כמו **df**, **du** ו-**mkfs** יציגו את גודל מערכת הקבצים ביחידות של בלוקים בני 512 בתים, כדי ליצור תאימות עם גרסאות מוקדמות של UNIX.

שטח תחזוקה	שטח ISL	רשימת בלוקים פגומים	שטח העברה להתקן השורש SWAP	בלוקים של נתונים	רשימת i	סופר בלוק	בלוק התיחול BOOT

תרשים 14.2 מבנה טיפוסי של מערכת קבצים

בלוק התיחול (boot block)

בכל מערכת קבצים בלוק אפס הינו בלוק התיחול. אין משתמשים בבלוק התיחול, אלא אם מערכת הקבצים היא מערכת הקבצים הראשית (root). בלוק התיחול מכיל פריטי מידע חשובים:

* מצביע לשטח התיחול (ראה שטח ISL).
* מצביע לשטח המשמש לתחזוקה.
* מידע התלוי בהתקן, כמו מידע על תהליך התיחול ומידע על צורת הארגון של הדיסק.

סופרבלוק

בכל מערכות קבצים "סופרבלוק" הינו תמיד בלוק מספר 1. ההכרזה על המבנה שלו נמצאת בקובץ הכותרת

/usr/include/sys/filsys.h

עותק של בלוק זה נמצא באופן קבוע בזיכרון ו-**init** משתמשת בו בכל 20 שניות כדי לעדכן את נתוני התפושה של הדיסק. הוא מכיל מידע גלובלי על מבנה מערכת הקבצים:

* גודל בבלוקים של רשימת i.
* גודל בבלוקים של כל הכרך (volume).
* רשימת צמתי i (inode) פנויים.
* רשימת בלוקים פנויים.
* שם מערכת הקבצים.

* דגל לציון מצב קריאה/כתיבה.
* ערך המציין את גודל הבלוק (512 או 1024 בתים).
* שם התקן לוגי.

רשימת I

רשימת I מכילה את מספר צמתי i (inodes) בדיסק. כל קובץ במערכת מזוהה על ידי מבנה ייחודי הנקרא צומת i (להלן inode). כל צומת כזה הוא בן 64 בתים. אם גודל הבלוק במערכת בה הוא 1024 בתים, הוא יכול להכיל 16 צמתים. ההכרזה על מבנה inode נעשית בקובץ כותרת בשפת C הנקרא "/usr/include/sys/ino.h", כפי שנראה להלן:

```
struct dinode
{
unsigned short di_mode;/* mode and type of file   */
short  di_nlink;       /* number of links to file */
unsigned short di_uid; /* owner's user id         */
unsigned short di_gid; /* owner's group id        */
long   di_size;        /* number of bytes in file */
char   di_addr[40]     /* disk block addresses    */
long   di_atime;       /* time last accessed      */
long   di_mtime;       /* time last modified      */
long   di_ctime;       /* time created            */
};                   /* of the 40 address bytes 39 used;
                        13 addresses of 3 bytes each  */
```

שלוש עשרה הכתובות מצביעות על הבלוקים של הנתונים בקובץ, אשר משתמשים בהן כפי שנפרט להלן. עשר הכתובות הראשונות, שנכנה אותן "בלוקים ישירים", מתייחסות לעשרת הבלוקים הראשונים של נתוני' בקובץ. אם הקובץ גדול יותר מעשרה בלוקים, תתייחס הכתובת האחת-עשרה לבלוק, שנכנה אותו "בלוק עקיף", שבו 256 כתובות לבלוקים נוספים של נתונים. הכתובת השתים-עשרה מצביעה על "בלוק עקיף כפול", אשר מכיל כתובות של 256 בלוקים עקיפים. הכתובת השלוש-עשרה מצביעה על "בלוק עקיף משולש", אשר מכיל כתובות של 256 בלוקים עקיפים כפולים.

אם גודל הבלוק במערכת הוא 1024 בתים, אז בלוק עקיף מכיל 256 מצביעים. במערכת שגודל הבלוק בה הוא 512 בתים, מכיל הבלוק הלא ישיר 128 מצביעים. מכאן שגודל הקובץ המקסימלי שניתן לנהל במערכות UNIX שגודל הבלוק בהן הוא 1024, או 512 בתים, הוא:

1024*(10+256+256*256+256*256*256)=17,247,250,432 bytes

או:

512*(10+128+128*128+128*128*128)=1,082,201,088 bytes

בלוקים של נתונים

בלוקים אלה מכילים את הנתונים של הקובץ אשר מסודרים במערך בתים חד-מימדי. כמה מבין בלוקים אלה משמשים את צמתי i כדי לאחסן מצביע יחיד, מצביע כפול או משולש. הבלוקים שאינם מוקצים לנתונים, לסופרבלוק ול-inodes הינם הבלוקים החופשיים.

מרכיבים נוספים של מערכת הקבצים

מערכת קבצים יכולה להכיל שטח העברה (swap area), רשימת בלוקים פגומים, שטח ISL ושטח תחזוקה.

שטח העברה, הקיים בהתקן הראשי (root) בלבד, משמש כשטח עבודה להעברת תכניות לתוך הזיכרון וממנו (ראה גם הסבר על הסיבית הצמודה).

רשימת הבלוקים הפגומים מתייחסת לבלוקים הפגומים בדיסק הקשיח. אם יתגלו בלוקים פגומים נוספים, ניתן להשתמש בפקודה **format** עם האופציה לתחזוקה, כדי להוסיף אותם לרשימה זו.

שטח ISL מיועד למטען המערכת (system loader), אשר המצביע שלו נמצא בבלוק המשמש לתיחול. ניתן למצוא את שטח ISL רק בהתקן הראשי והוא מכיל את המטען של "unix/" לתוך הזיכרון.

מהנדסי תוכנה מריצים לעתים תכניות בדיקה מסוימות כדי לבחון את המערכת. השטח המיועד לתחזוקה בדיסק הקשיח משמש לתכניות אלה כשטח עבודה.

14.9 יצירת מערכת קבצים חדשה

ניתן ליצור מערכת קבצים חדשה בעזרת הפקודה **mkfs**, אשר נמצאת בדרך כלל במדריך "etc/". כאשר יוצרים מערכות קבצים חדשות, הן תופענה כרשומות מדריך במדריך הראשי (root). לדוגמה, כדי ליצור מערכת קבצים בת 1500 בלוקים בתקליטון ששמו "dev/fd50/", נכתוב את הפקודה:

```
$/etc/mkfs /dev/fd50 1500
```

14.10 תחזוקה של מערכת הקבצים

בשלב עבודה כלשהו יכולה מערכת הקבצים להפגם מסיבות של תקלות חומרה, כיבוי המערכת ללא ביצוע **sync**, או טעויות של משתמש-על. מערכת קבצים נחשבת כפגומה, כאשר לדוגמה, מתרחשים אחד או יותר מן הדברים הבאים:

* שני צמתי i (inodes) מתייחסים לאותו בלוק נתונים.
* ישנו בלוק פגום, מכיון שצומת i מתייחס למספר בלוק הנמצא מחוץ לתחום שבין בלוק הנתונים הראשון לבין הבלוק האחרון בהתקן הלוגי.
* בלוק מסוים מוגדר כשייך לקובץ, וגם נמצא ברשימת הבלוקים הפנויים.

קיימות מספר תכניות לתיקון מערכת קבצים פגומה. תכניות אלה הן **check**, **ncheck**, **dcheck**, **icheck**, **fcheck**, **fsdb** ו-**fsck**.

תיקון אינטראקטיבי של מערכת הקבצים

הפקודה **fsck** (file system check) מאפשרת תיקון אינטראקטיבי של מערכת הקבצים. מערכת הקבצים הראשית הינה ברירת מחדל בפקודה זו, אך ניתן לבדוק מערכות קבצים אחרות, אם נרשום אותן בקובץ "/etc/checklist".

הפקודה **fsck** תעבור על כל מערכת קבצים בתורה, תדווח על חריגות ותבקש מהמשתמש להגיב בהתאם: 'y' ("כן") כדי לתקן את הבעיה ו-'n' ("לא") כדי לא לנקוט בכל פעולה שהיא. הרצת **fsck** עם האופציה **n-** תגרום לתשובה שלילית אוטומטית ותפיק רשימה של כל החריגות. ניתן להשתמש ברשימה זו לאחר מכן עם תכניות שירות שונות, כמו **ncheck** ו-**fsdb** כדי לתקן את מערכת הקבצים באופן ידני. באופן דומה, הרצת **fsck** עם האופציה **y-** תגרום לתשובה חיובית אוטומטית שלא תדרוש כל התערבות נוספת מצד המשתמש.

לפני שמריצים את **fsck** חשוב לבדוק אם המדריך "/lost+found" קיים במערכת. המדריך "/lost+found" חייב להיות גדול דיו כדי לאחסן קבצים "יתומים" שהפקודה **fsck** תעתיק לתוכו. ניתן להבטיח זאת על ידי העתקת מספר קבצים גדולים ל-"/lost+found" וביטולם לאחר מכן.

לאחר שמערכת קבצים תוקנה בעזרת "fsck", יש לאתחל (reboot) את המערכת מבלי לבצע את פעולת **sync**. דבר זה יבטיח שכל תיקון שנעשה על ידי **fsck** בדיסק לא יבוטל על ידי העתקה של

נתונים לא נכונים ממערכת הקבצים המקורית (הפגומה) שהוטענה קודם לכן לזיכרון המרכזי. להלן תהליך טיפוסי של **fsck**:

```
$fsck
/dev/0s1
** Phase 1 - Check Blocks and Sizes
POSSIBLE FILE SIZE ERROR I = 3405
nblks=0, filesize=1
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
UNREF FILE I = 3405 OWNER = root MODE = 1
SIZE=0 MTIM=NOV 11 16:50 1988 -- CLEARED FIX ? y
** Phase 5 - Check Free List
5 BLK(S) MISSING
BAD FREE LIST
SALVAGE ? y
** Phase 6 - Salvage Free List
3426 files 54050 blocks 13068 free
*** BOOT THE SYSTEM - DO NOT PERFORM A SYNC OPERATION ***
```

בנוסף לאופציות **y**, **n**, **n-** ו-**y-** ישנה גם האופציה **s-**, אשר ניתן להשתמש בה כדי לבנות מחדש את רשימה הבלוקים הפנויים (ראה שלב 5 ושלב 6 בדוגמה לעיל). לדוגמה, הפקודה הבאה תבנה מחדש את רשימת הבלוקים הפנויים של "/dev/0s1":

```
$fsck -s /dev/0s1
```

ניתן להשתמש באופציה **s-** במצב שבו בלוקים, אשר רשומים כשייכים לקבצים קיימים נמצאים גם ברשימת הבלוקים הפנויים, או במצב שבו בלוקים פנויים, אינם נמצאים ברשימת הפנויים.

תיקון ידני של מערכת הקבצים

תיקון ידני של מערכת הקבצים יכול להעשות בעזרת הפקודות **fsck** ו-**ncheck**. להלן תהליך טיפוסי כזה.

הפעל את "fsck -n" על מערכת הקבצים (בדוגמה זו "/dev/0s1"), כדי לקבוע אילו בלוקים פגומים ואילו מהם כפולים:

```
$fsck -n /dev/0s1
36  BAD I = 98
102 DUP I = 1056
102 DUP I = 1072
```

בדרך זו הפקודה **fsck** מוצאת שיש בלוק אחד פגום ושני בלוקים כפולים. כעת יש למצוא את שם הקובץ הקשור לבלוק הפגום:

```
$ncheck -i 98 /dev/0s1
98 /usr/anne/swap.c
```

קושרים את הקובץ למדריך חדש:

```
$mkdir /usr/badblocks
$ln /usr/anne/swap.c /usr/badblocks/swap98
```

לאחר ביטול הקובץ המקורי עם הבלוק הפגום, מעתיקים את הקובץ בשמו המקורי:

```
$rm /usr/anne/swap.c
$cp /usr/badblocks/swap98 /usr/anne/swap.c
```

כפי שניתן לראות מביצוע הפקודה **"fsck -n"** ישנן גם שגיאות של כפילות בין בלוקים, כאשר שני **inodes** שונים (1056 ו-1072) טוענים לבעלות על אותו מספר בלוק. הנוהל שאומץ במקרה זה (כפי שנראה להלן) מורה שיש לשמור את שני הקבצים הטוענים לבעלות על אותו בלוק ולאחר מכן לשנות את שמותיהם:

```
$ncheck -i 1056 1072 /dev/0s1
1056 /usr/joy/master.c
1072 /usr/andreas/menu.c

$mkdir /usr/dupblocks
$cp /usr/joy/master.c /usr/dupblocks/master.c
$cp /usr/andreas/menu.c /usr/dupblocks/menu.c
$rm /usr/joy/master.c /usr/andreas/menu.c
$mv /usr/dupblocks/master.c /usr/joy/master.c
$mv /usr/dupblocks/menu.c /usr/andreas/menu.c
```

עכשיו הרץ שוב את הפקודה **"fsck -n"** כדי לוודא שאין חריגות נוספות, או חוסר עקביות, במערכת הקבצים.

ניקוי חירום של מערכת הקבצים

במקרים נדירים מאד יתגלה בלוק פגום ברשימת צמתי i ‏(inode). בעיה זו תגרום לכך שיוקצו מספרי בלוקים לא חוקיים לצומת i נתון. במקרה זה יש לבטל את הצמתים ולכלול את הבלוק ברשימת הבלוקים הפגומים על ידי שימוש באופצית התחזוקה של פקודת **format**. ניתן לבטל צומת i על ידי שימוש בפקודה

```
$/etc/clri filesystem inumber
```

"filesystem" הוא שם של קובץ מסוים, או של מערכת הקבצים, שיש לתקן אותם ו-"inumber" הינו מספר צומת **i** שיש לבטל.

כהערה מסכמת, יש לזכור לעשות את הדברים הבאים:

* לאתחל (reboot) את המערכת לאחר תיקון מערכת הקבצים הראשית.
* לטעון את מערכת הקבצים לאחר שבוצעו התיקונים הדרושים בקבצים הניתנים להסרה.

14.11 הוספת התקנים חדשים למערכת UNIX

כדי להוסיף התקן (device) חדש למערכת, יש להשתמש תחילה בפקודה **mknod** כדי לרשום את שם ההתקן במדריך "/dev". לדוגמה, כדי להוסיף שם התקן חדש עבור מסוף "tty08" יש להשתמש בפקודה הבאה:

```
$/etc/mknod /dev/tty08 c min maj
```

הארגומנט השני של הפקודה, מציין אם ההתקן הוא התקן תווים (**c**) או התקן בלוקים (**b**). בדוגמה זו מגדירים התקן תווים. הארגומנטים השלישי והרביעי, **min** ו-**maj**, מציינים את מספרי ההתקן המשני והעיקרי בהתאמה (ראה פרק 2). יש לקבל מספרים אלה מיצרן המערכת. לאחר יצירת הקובץ המיוחד עבור המסוף, יש לנקוט בפעולות הבאות:

* הכן רשומה בקובץ "/etc/gettydefs", כדי לתאר את מאפייני הקו שאליו יחובר המסוף. להלן רשומה טיפוסית:

```
d# B9600 CS8 ECHO BRKINT IGNPAR ICANON IXON IXOFF
         CLOCAL TAB3 ISIG OPOST
```

* הכן את הרשומות הדרושות בקובץ "/etc/termcap", אשר מתארות את מאפייני המסוף.

* הכן רשומה בקובץ "/etc/inittab", אשר תתחיל את "getty" כאשר התהליך סיים את פעולתו, או כאשר המשתמש יצא מהמערכת. רשומה טיפוסית:

```
t08:1:respawn:/etc/getty tty08 d wys85
```

* השתמש בפקודה הבאה:

```
$/etc/init q
```

היא תגרום לכך שתהליך **init** יקרא מחדש את קובץ "/etc/inittab" הכולל עתה את השינויים שנעשו בו.

פרק 15

גישה למידע אודות המערכת

הפקודה **ps** מספקת מידע על מצב התהליכים הפעילים במערכת. המידע המדווח לאחר ביצוע פקודה זו מספיק ברוב המקרים, אך במקרים מסוימים נדרש מידע נוסף. בפרק זה נראה, באמצעות תכנית בשפת C (תכנית "pstat.c"), כיצד ניתן להשיג מידע נוסף על מצב התהליכים הפעילים במערכת.

כאשר קובץ ביצוע נטען לזיכרון, נקבעים בו שלושה סגמנטים לוגיים: סגמנט הטקסט (הפקודות), סגמנט הנתונים וסגמנט המחסנית. הסבר מפורט על סגמנטים אלה ועל מבנה קובץ יעד משותף ניתן בפרק 8.

התכנית "pstat.c" נותנת, בנוסף למידע אחר, את גודל הזיכרון בבתים, שהוקצה לסגמנטים של הטקסט (הפקודות), הנתונים והמחסנית השייכים לכל תהליך פעיל במערכת. מידע זה מועיל מאד בזמן פיתוח תוכנה ובזמן הסבת תוכנה ממערכות הפעלה אחרות ל-UNIX.

15.1 הקבצים שהתכנית משתמשת בהם

התכנית המודגמת "pstat.c" משתמשת בשני קבצים: קובץ **mem**, הנמצא במדריך **dev/** וקובץ **unix**, הנמצא במדריך הראשי.

קובץ **dev/mem/** שייך לקטגוריית קבצים/התקנים מיוחדים ב-UNIX. זהו התקן תווים טורי המתייחס לזיכרון שבשליטת התכניות הנוכחיות. מסיבות ביטחון יש לשנות את שמות הבעלים והקבוצה של כל תכנית הפונה לקובץ זה ולהדליק בה את סיבית הזיהוי של המשתמש (SUID).

קובץ **unix/** נוצר על ידי הידור וקישור של קבצי המקור של UNIX ומצורפות לו טבלאות ניוד עבור הגרעין. פקודות כמו **ps** והתכנית "pstat" יסרקו את הטבלאות האלו כדי למצוא, לדוגמה, את המיקום של טבלאות העיבוד ועל ידי כך להשיג מידע על מצב התהליכים הפעילים במערכת. הקובץ **unix/** נטען לזיכרון במהלך התיחול.

15.2 המבנים שהתכנית משתמשת בהם

התכנית "pstat" משתמשת במבנים **LDFILE**, **SAYMENT** ו-**proc**, אשר נמצאים בקבצים "idfch.h", "syms.h" ו-"proc.h" בהתאמה.

המבנה "LDFILE" משמש לאחסון מידע על סוג הקובץ שבשימוש, על כותרת הקובץ הפתוח ועל הכתובת היחסית המוחלטת של תחילת הקובץ (ראה שגרת **ldopen** בסעיף הבא).
המבנה "SAYMENT" משמש לאחסון ערך של סמל, מספר הסעיף וכתובת יחסית בטבלת המחרוזות של קובץ היעד.

לכל תהליך פעיל במערכת מוקצה מבנה "**proc**" אחד. מבנה זה משמש לאחסון מידע כמו עדיפות, מספר הזיהוי האמיתי של המשתמש, מספר הזיהוי של תהליך האב, גודל הסגמנטים המשמשים את הטקסט והמחסנית וגודל הסגמנטים של הנתונים.

15.3 השגרות שהתכנית משתמשת בהם

התכנית **pstat** משתמשת בשגרות המערכת הבאות:

ldopen משמשת לפתיחת קובץ יעד לקריאה ו-**ldclose** משמשת לסגירת קובץ כזה. **ldopen** מתוכננת לאפשר גישה לקבצי היעד המוגדרים כארכיב. היא מקצה ומאתחלת את המבנה "LDFILE" ומחזירה לתכנית שקראה לה את המצביע למבנה זה.

ldtbseek מחפשת מקום מסוים בטבלת הסימנים של קובץ היעד. היא מחזירה SUCCESS או FAILURE.

ldtbindex מחזירה את האינדקס של רשומה הנמצאת בטבלת הסימנים בקובץ יעד משותף. לאחר מכן ניתן להשתמש באינדקס זה בקריאות לשגרה "ldtbread".

ldtbread פקודת השגרה:

ldtbread(pld,sindx,sysmbol)

השגרה קוראת מטבלת הסימנים רשומה, שצויינה על ידי "sindx", ומעבירה אותה לשטח הזיכרון המתחיל ב-"symbol" של קובץ היעד הקשור ל-"pld". היא מחזירה SUCCESS או FAILURE.

ldgetname משמשת לאחזור השם הקשור לרשומה מסוימת בטבלת הסימנים שהתמלאה כתוצאה מקריאה מוצלחת לשגרה "ldtbread".

15.4 תהליך ההכנה

סעיף זה מתאר כיצד יש להדר ולקשר את התכנית "pstat" וכיצד לעשותה זמינה לכל המשתמשים במערכת.

תחילה צריך לבצע הידור של התכנית עם האופציה **-lld**, מכיון שיש לטעון את התכנית עם קובץ היעד של ספריית שגרות הכניסה "libld.a":

```
$cc -o pstat pstat.c -lld
```

בהנחה שאין שגיאות תחביריות, יופק הקובץ ביצוע "pstat". חשוב עתה לקבוע את הערכים המתאימים של הבעלים, את הקבוצה ואת ה-mode של "pstat", כדי לאפשר לה גישה לקובץ "/dev/mem". הפקודה "ls -l" הפועלת על הקבצים "pstat" ו-"/dev/mem" תפיק לדוגמה, את הפלט הבא:

```
$ls -l pstat /dev/mem
-rwxr-xr-x 1 andy    others 38025 Jan 10 11:30 pstat
crw-r----- 1 sys     sys     1,  0 Jun 15 12:30 /dev/mem
```

כדי לגשת לקובץ "/dev/mem" צריך להכנס למערכת (login) כשורש. במצב זה ניתן לשנות את זיהוי הבעלים וזיהוי הקבוצה של "pstat" ל-"sys" ולהדליק את סיבית הזיהוי של המשתמש (SUID):

```
$chown  sys  pstat
$chgrp  sys  pstat
$chmod  u+s  pstat
$ls -l pstat /dev/mem
-rwsr-xr-x 1 sys     sys     38025 Jan 10 11:33 pstat
crw-r----- 1 sys     sys     1,  0 Jun 15 12:30 /dev/mem
```

לבסוף, צריך להעביר את התכנית "pstat" למדריך "/usr/bin" כדי שכל המשתמשים במערכת יוכלו לפנות אליה.

15.5 התכנית pstat

בסעיף זה נציג את תכנית המקור "pstat.c".

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>
#include <sys/types.h>
#include <sys/proc.h>
#include <sys/param.h>
#include <sys/sysmacros.h>

main()
{
long szproc, adproc;
int flag;

/* locate proc table and get its size and address */

flag=findproc(&szproc,&adproc);
/*search the proc table for processes and print information */
if( !flag )
     flag=lookproc(szproc,adproc);
if( !flag )
     exit(0); /* to indicate no error */
exit(1);
}

/*Routine       : To locate proc table and get its size
                  and address */

static int findproc(szproc,adproc)
long *szproc,*adproc;
{
long lindx;
extern long ldtbindex();
extern int ldtbseek(), ldtbread();
extern char *ldgetname();
SYMENT symbol;
int f12,f13;
LDFILE *pld, *ldopen();
char *ldret="A*A*A*A*A*A*A*A*A*A*A*A*A*A*A*";
```

```
if( ( pld=ldopen("/unix",pld) ) == NULL )
     {
     fprintf(stderr,"\nCannot open /unix file\n");
     return(1);
     }

if( ldtbseek(pld) == FAILURE )
     {
     fprintf(stderr,"\nCannot seek to symbol table of /unix\n");
    return(1);
     }
fl2=SUCCESS;
fl3=1;
while ( fl2==SUCCESS && fl3 )
     {
     lindx=ldtbindex(pld);
     fl2=ldtbread(pld,lindx, &symbol);
     /* locate the name "proc" */
     ldret=ldgetname(pld, &symbol);

     /* check if "proc" has been located */
     fl3=strcmp("proc", ldret);
     }
if( !fl3 )
     *adproc=symbol.n_value; /* "proc" has been located */
else
     {
     fprintf(stderr,"\nproc name cannot be located\n);
     return(1);
     }
ldtbread(pld, ldtbindex(pld), &symbol);
*szproc = symbol.n_value - 1;
ldclose(pld);
return(0);
}
```

```
/* Routine      : To search the proc table for processes and
                  initiate printing of process information */

static int lookproc(szproc, adproc)
long szproc, adproc;
{
int fd;
FILE *pfd, *popen();
void h_print(), r_line(), p_line();
char uid[12],tty[12],com[12],pid[12],ppid[12],time[12],stime[12];
char cline[80];
struct proc buf;

if( ( fd=open("/dev/mem",0) ) == -1 )
     {
     fprintf(stderr,"\nCannot open the /dev/mem file \n");
     return(1);
     }

 h_print(); /* print heading */
 /* use a pipe with the sed command to get output from the second line */
 /* of the "ps" command onwards */
 pfd=popen( "ps -ef | sed -n '2,$p'", "r" );
 while ( szproc > adproc )
      {
|
  lseek(fd,adproc,0);
  read(fd,&buf,sizeof(buf));
  if(buf.p_stat != 0 )
       {
       fscanf(pfd,"%s %s %s %s %s %s %s",uid,pid,ppid,com,stime,
              tty,time);
       /* read the command from the pipe */
       r_line(cline,pfd);
       /* output all information about the process now */
       p_line(uid,tty,cline,pid,ppid,&buf);
       }
   adproc += sizeof(buf);
   }
```

```
printf("|------------------------------------------------------------|\n);
pclose(pfd); /* close pipe */
close(fd);   /* close /dev/mem */
return(0);
}

/* Routine     : To print the heading of output */
static void h_print()
{
printf("| login | device | command | proc | parent | text | data|
        stack |\n");
printf("|       |        |         |      |        | (KB) | (KB)|
        (KB)  |\n");
printf("|------------------------------------------------------------|\n);
}

/* Routine     : To output process status information */
static void p_line(uid, tty, cline, pid, ppid, buf)
char *uid, *tty, *cline, *pid, *ppid;
struct proc *buf;
{
/* "ctob" converts number of characters to blocks of size
   1024 bytes each , as shown in the code below */
printf("| %7.7s | %7.7s | %15.15s | %7.7s | %6.6s | %5d | %4d |%4d |\n",
         uid,tty,cline,pid,ppid,
        ctob(buf->p_tsize)/1024, ctob(buf->p_size)/1024,
        ctob(buf->p_ssize)/1024);
}

/* Routine     : To read a line from the process status pipe */
static void r_line(cline,pfd)
char *cline;
FILE *pfd;
{
     while ( ( *cline++ = getc(pfd) ) != '\n' );
     *--cline='\0';
}
```

15.6 הפלט של התכנית

להלן פלט אפשרי מהתכנית "pstat".

login	device	command	proc	parent	text (KB)	data (KB)	stack (KB)
root	10	37:18 sw	0	0	0	4	4
root	10	0:03 /e	1	0	32	24	12
root	10	0:03 sw	0	0	0	4	4
root	console	/etc/getty con	74	1	32	24	12
root	tty02	/usr/getty tty	75	1	32	24	12
root	console	/usr/lib/errde	58	1	12	20	12
root	?	/etc/cron	68	1	32	28	12
lp	?	/usr/lib/lpsch	72	1	36	28	12
andy	tty10	-sh	77	1	44	32	12
andy	tty11	-sh	78	1	44	32	12
root	tty19	/etc/getty tty	79	1	32	24	12
andy	tty11	cc master.c	175	78	20	20	12
andy	tty10	vi subs.c	100	77	112	72	20
andy	tty11	as -o master.o	179	175	48	224	12
andy	tty11	pstat	178	78	20	24	12
andy	tty11	[sh]	180	178	0	0	48

נספח א'

התווים השמורים של המעטפת

רשימת התווים השמורים (metacharacters) של המעטפת:

<table>
<tr><td>></td><td>ניתוב פלט.</td></tr>
<tr><td>>></td><td>הוספת פלט.</td></tr>
<tr><td><</td><td>ניתוב קלט.</td></tr>
<tr><td><<</td><td>קלט ממסמך here.</td></tr>
<tr><td>|</td><td>סימן הצינור.</td></tr>
<tr><td>&&</td><td>סימן "and if" (למשל a&&b: הרץ את a ובמקרה של הצלחה, הרץ את b).</td></tr>
<tr><td>||</td><td>הסימן "or if" (למשל, a||b: הרץ את a ובמקרה של כישלון הרץ את b).</td></tr>
<tr><td>;;</td><td>מגביל גודל אות.</td></tr>
<tr><td>;</td><td>מפריד בין פקודות.</td></tr>
<tr><td>'..'</td><td>החלפת פקודות.</td></tr>
<tr><td>()</td><td>קיבוץ פקודות והרצה שלהן בתת-מעטפת.</td></tr>
<tr><td>*</td><td>תואם לכל צירוף של תווים.</td></tr>
<tr><td></td><td>תואם לכל תו יחיד.</td></tr>
<tr><td>[..]</td><td>תואם לכל התווים שבסוגריים.</td></tr>
<tr><td>\</td><td>התייחס לתו הבא כלשונו (ולא כסימן או הוראה).</td></tr>
<tr><td>".."</td><td>הצג את התווים שבין המרכאות הכפולות, פרט לתווים $,",',/.</td></tr>
<tr><td>'..'</td><td>הצג את התווים שבין סימני הטעם פרט לסימן '-'.</td></tr>
<tr><td>$0</td><td>שם קובץ פקודות המעטפת הפעיל (הקורא).</td></tr>
<tr><td>$1..$9</td><td>ארגומנטים לקובץ פקודות המעטפת.</td></tr>
<tr><td>$name</td><td>ערך של שם משתנה.</td></tr>
<tr><td>${name}</td><td>החלפה של שם משתנה.</td></tr>
</table>

נספח ב'

תבניות תכנות במעטפת בורן ובמעטפת C

טבלת השוואה בין תבניות התכנות בשתי המעטפות.

Bourne shell	C shell
if condition then statements elif condition then statements else statements fi	if (condition) then statement(s) else if (condition) then statements else statements endif
times	time
for name [in word-list] do statement(s) done	foreach name [word-list] statement(s) end
case word in option) cmds;; *) cmds;; esac	switch (word) case option: cmds breaksw default: cmds breaksw endsw
exit [expr]	exit [(expr)]
export	setenv
trap 'cmds' signals	onintr label
. filename	source filename
continue	continue
while condition do statements done	while (condition) statements end
read	---
readonly	---
break [n]	break [break]

נספח ג'

טבלאות ASCII

הטבלאות מציגות את הערכים של כל תו ASCII לפי בסיס 16 ולפי בסיס 8.

טבלה ג-1 הערכים של תווי ASCII לפי בסיס 8 (אוקטלי)

000 nul	001 sch	002 stx	003 etx	004 ect	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dcl	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 $	045 %	046 &	047 '
050 (	051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [	134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174 |	175 }	176 ~	177 del

טבלה ג-2 הערכים של תווי ASCII לפי בסיס 16 (הקסדצימלי)

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dcl	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

ביבליוגרפיה

מאמרים שונים פורסמו בתקופונים, בעיתונות המקצועית ובעיתונות הכללית.

- אנשים ומחשבים.
- מחשבים / מירב תעשיות הפקה.
- מידעון / איגוד מנתחי מערכות לענ"א.
- מעשה חושב / איל"א.
- רשת מחשבים / מירב תעשיות הפקה.

פרסומים בעברית

- קטלוג מוצרים למערכת הפעלה SCO XENIX/UNIX.
המדריך ליישומים, תוכנה בסיסית וחומרה.
יוניפאור, יוני 1989.

ספרים ופרסומים באנגלית

Gircys, Gintaras R./UNDERSTANDING AND USING COFF./ O'Reilly & Associates, Inc.
Explains the Common Object File Format used in UNIX System V, along with the related behavior of programs that use this format, such as the UNIX assembler and linker.

Groff J. & Weinberg p./UNDERSTANDING UNIX/ QUE Publishing, 1988.

Libes, Don and Ressler, Sandy/LIFE WITH UNIX: A GUIDE FOR EVERYONE/Prentice-Hall, 1989.
Presents a practical guide to working with UNIX, with sections on UNIX networks, services, applications software, hardware implementations, and tips for users, programmers, and administrators.

Pilavakis, A.J./UNIX Workshop/ Macmillan England, 1989.

UNIX FOR BEGINNERS/ Abacus, 1989.
Describes the popular Unix system from the user's point of view, discussing the logon procedures, file concepts, and commands.

UNIX SYSTEM SOFTWARE READINGS/ Prentice-Hall, 1988.
Presents the proceedings of AT&T Unix Pacific Co., Ltd.'s UNIX system Software Technology Seminar for the Asia/Pacific region held in July 1986 that featured six speakers whose expertise covers computer science research, system architecture, and system development of UNIX system V.

Waite, Martin & Prata/ UNIX Primer Plus (2nd Ed)/ Howard W. Sams & Co., 1990.
How to master **ed, vi, emacs** editors; Interact with UNIX shells; Set up UNIX's advanced mail features; Discover redirection and background/foreground operations; Customize your computer with UNIX's advanced editing techniques.

Periodicals

- ComputerWorld
- UNIX World
- UNIX Review

אינדקס

האינדקס בנוי לפי סדר המיון של המונחים באנגלית. במידת האפשר ניתן תרגום או הסבר של המונח ומציאת מקום המונח בספר. ציינו בסוגרים כאשר המונח מתייחס לגרעין UNIX או לשפת C.

ללמוד UNIX פונה אל המשתמשים ב-UNIX בתחום העסקי והאקדמי ואל כל אלה, אשר מתכוונים לפתח יישומים ומערכות או להגר אל סביבת עבודה זו.

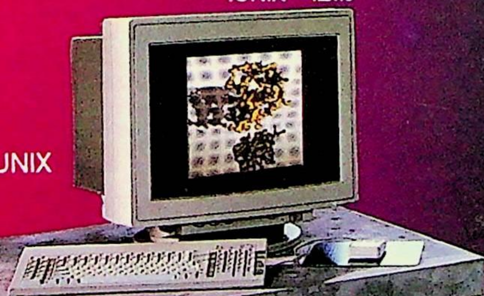
עבור המתחילים, המצטרפים לעולם UNIX, המחבר מציג את עקרונות המבנה של מערכת ההפעלה, הפקודות הבסיסיות הנחוצות להם בתחילת עבודתם, מערכת ניהול הקבצים, המעטפת (Shell) והעורך vi.

למתקדמים ולמי שכבר עוסק בפיתוח יישומים, המחבר מסביר מרכיבי שירות נוספים: הקמת פרויקטים, שגרות שירות של make ושל sccs, עבודה בשפת C, מעבד הטכסט nroff וקשר בין תהליכים פנימיים באמצעות אתתים ושיתוף זיכרון.

לאנשי התחזוקה של מערכות UNIX המחבר מציע כלים לשליטה וניהול, קישור מערכות באמצעות uccp, דואר אלקטרוני, ניהול גיבויים, חיבור התקני קלט/פלט חדשים, הוספת משתמשים ותיקון תקלות במערכת הקבצים.

בכל נושא שהמחבר דן בו, הוא מרכז את המידע הרלוונטי ומציג אותו בבהירות ובפשטות ברמה הדרושה והעיקר - מלווה את ההסברים **בדוגמאות רבות המפרטות את הפקודות ואת הפלט.**

אנדריאס פילאווקיס סיים את לימודיו באוניברסיטאות לונדון (MSc) במדעי המחשב ופיתוח מערכות מידע עסקיות. בין עיסוקיו המגוונים: מנהל פרויקטים, יועץ, מדריך ומנתח מערכות בסביבת עבודה של UNIX בארגונים עסקיים שונים ובחברות שיווק של מוצרי UNIX.



מסת"ב ISBN 965-361-011-2